Trabajo Fin de Máster
Máster en Ingeniería Industrial

Diseño, montaje y puesta en marcha de vehículos aéreos multirrotor para su uso en redes de comunicaciones aéreas

Autor: Sergio Vigorra Treviño
Tutor: Daniel Gutiérrez Reina
Cotutor: Jesús Sánchez García

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Diseño, montaje y puesta en marcha de vehículos aéreos multirrotor para su uso en redes de comunicaciones aéreas

Autor:
Sergio Vigorra Treviño

Tutor:
Daniel Gutiérrez Reina
Profesor sustituto

Cotutor:
Jesús Sánchez García

Depto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Máster: Diseño, montaje y puesta en marcha de vehículos aéreos multirrotor para su uso en redes de comunicaciones aéreas

Autor: Sergio Vigorra Treviño

Tutor: Daniel Gutiérrez Reina

Cotutor: Jesús Sánchez García

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

Sobre el autor

Sergio Vigorra Treviño

svigorra@gmail.com

Máster en Ingeniería Industrial

Universidad de Sevilla

Agradecimientos

La realización de este proyecto supone el final (salvo sorpresas) de mi formación académica como Ingeniero Industrial. Su concepción se produjo en un momento en el que personalmente me sentía especialmente desencantado con el sistema universitario, con su estructura y con algunas de las personas que lo componen. Fue entonces cuando tuve la suerte de conocer a Federico, profesor que imparte una asignatura muy estimulante, y junto con él plantear un proyecto que abarcara esta temática (la cual me encanta) y poder disponer de medios materiales para hacer una implementación real.

Posteriormente pude conocer Jesús y Daniel, excelentes personas a las que apasiona su trabajo y con las que he tenido la suerte de poder trabajar durante meses. Agradezco especialmente su total disponibilidad, su interés por el proyecto y todo el apoyo que han podido darme. En este punto quisiera mencionar también a Pepe y Alberto, quienes sin tener ninguna obligación se han interesado por aprender lo que he podido enseñarles del mundo de los multirrotores. Espero que algún día puedan continuar de algún modo con el trabajo que he comenzado yo.

Por supuesto tengo que mencionar en este capítulo de agradecimientos, con mucho orgullo, a mi familia, a mi novia y a mis amigos, ya que decir que me han dado su apoyo continuamente sería quedarme muy corto.

Para finalizar, quisiera hacer una mención a los movimientos de hardware y software libre y a las personas que soportan los distintos proyectos de este tipo y contribuyen a distribuir el conocimiento tecnológico. Sirva como ejemplo la plataforma *Ardupilot*, en la cual están basados los desarrollos realizados en este proyecto.

Sergio Vigorra Treviño

Sevilla, 2016

Resumen

Los multirrotores son un tipo de aeronaves no tripuladas las cuales han experimentado un gran auge en los últimos años y para los cuales cada vez son más numerosas las aplicaciones desarrolladas.

El presente trabajo consiste en una aproximación hacia la creación de redes de comunicaciones aéreas basadas en este tipo de vehículos. En lo que respecta a los multirrotores, el documento abarca la descripción teórica de los vehículos y de los distintos sistemas que lo componen, la selección de componentes, el montaje y la puesta a punto. Posteriormente se explican una serie de desarrollos software llevados a cabo para implementar comunicación entre multirrotores.

El sistema descrito consiste en añadir a cada multirrotor un ordenador de placa única tipo Raspbery Pi así como un módulo Wi-Fi. Sobre este hardware se han desarrollado aplicaciones en lenguaje Python para acceder a los datos de navegación del autopiloto y compartirlos con otros multirrotores.

Abstract

Multirotor*s* are a type of unmanned aircraft which have experienced a significant growth in recent years, for which there is also an increasing number of applications.

The present study approaches the creation of aircraft networks based on this type of vehicle. Concerning multirotor*s,* this document covers the theoretical description of the vehicles and the different systems that make them up, as well as the selection of components, the assembly and the adjustment. Subsequently, a series of software developments made in order to implement communication among multirotors are explained.

The described system consists in adding to every multirotor a Raspberry Pi single board computer, as well as a Wi-Fi module. Different Python language based applications have been developed in relation with this hardware in order to access navigational information of the autopilot and share it with other multirotors.

Índice

Índice de Figuras

Índice de Tablas

Notación

ALO	Avión Ligero de Observación
APM	*AutoPilotMega*
BEC	*Battery Eliminator Circuit* o circuito eliminador de batería
DC	*Direct Current* o corriente continua
ESC	*Electronic Speed Controller* o variador electrónico de velocidad
FPV	*First Person View* o vista en primera persona
GCS	*Ground Control Station* o estación de control en tierra
GPS	*Global Positioning System* o sistema de posicionamiento global
IMU	*Inertial Measurement Unit* o unidad de medición inercial
INTA	Instituto Nacional de Técnica Aeronáutica
MTOW	*Maximum Take Off Weight* o peso máximo al despegue
OSD	*On Screen Display* o display en pantalla
PCB	*Printed Circuit Board* o placa de circuito impreso
PDB	*Power Distribution Board* o placa de distribución de potencia
PPM	*Pulse Position Modulation* o modulación por posición de pulso
PWM	*Pulse Width Modulation* o modulación por ancho de pulso
R/C	Radiocontrolado
RDP	*Remote Desktop Protocol* o protocolo de escritorio remoto
RPA	*Remotely Piloted Aircraft* o aeronave pilotada remotamente
RPI	*Raspberry Pi*
UA	*Unmanned Aircraft* o aeronave no tripulada
UAS	*Unmanned Aircraft System* o sistema de aeronave no tripulada
UAV	*Unmanned Aircraft Vehicle* o vehículo aéreo no tripulado
UCAV	*Unmanned Combat Aircraft Vehicle* o vehículo aéreo de combate no tripulado

1 INTRODUCCIÓN Y OBJETIVOS

El fracaso es una opción. Si las cosas no están fallando,
no estás innovando lo suficiente.

- Elon Musk -

Durante los últimos años los vehículos aéreos no tripulados, también conocidos comúnmente como drones o Unmanned Aerial Vehicles (UAV), han experimentado un notable desarrollo llegando a cubrir actividades que tradicionalmente llevaba a cabo la aviación convencional (tripulada). Hasta hace no demasiado tiempo para poner en vuelo algún tipo de aeromodelo no tripulado eran necesarios unos conocimientos y destrezas muy concretos. Principalmente ha sido el desarrollo de la electrónica de control el que ha propiciado que hoy en día esto sea mucho más sencillo.

El mercado ha experimentado una apertura importante, siendo ahora muchas las empresas que comercializan aeronaves no tripuladas listas para volar o que ofrecen sus servicios para realizar actividades como por ejemplo la fotografía aérea.

De entre todos los tipos de aeronaves no tripuladas, los multirrotores han sido los que más éxito han experimentado, debido principalmente a su simplicidad mecánica y su versatilidad, que se manifiesta en la posibilidad de despegar/aterrizar verticalmente y de hacer vuelo estacionario, combinado con una capacidad de carga y autonomía de vuelo aceptables para muchas aplicaciones.

El objetivo de este proyecto será estudiar el guiado autónomo de varios multirrotores, así como las comunicaciones establecidas entre ellos. La meta final que se propone en este proyecto es implementar un prototipo de red aérea compuesta por dos multirrotores.

1.1 Concepto

Los UAVs son una herramienta muy potente y versátil, y tienen una serie de ventajas que los convierte en una opción muy interesante para muchas aplicaciones ya existentes además de para otras que han sido ideadas *ex profeso* a raíz de su existencia. Además, el auge de estos dispositivos ha sido tan reciente que, por ejemplo, la legislación vigente en España aún no es definitiva (BOE 252 de 17 de octubre de 2014, Artículo 50: *Operación de aeronaves civiles piotadas por control remoto*).

Uno de los desafíos para el futuro es el desarrollo de aplicaciones colaborativas de un cierto número de UAVs, de manera que éstos, guiados autónomamente y sin control humano directo, puedan realizar operaciones tales como recopilar datos, compartir información entre sí y con otros elementos del entorno, realizar correcciones de rumbo en función de la información recopilada, etc.

Este tipo de gestión inteligente del vuelo y comunicación con el entorno es determinante para la integración de una flota de UAVs dentro de un sistema mayor, como podría ser un sistema automatizado de transporte de material, sistema de control/monitorización del tráfico rodado, monitorización de zonas de desastre, etc.

Es en este ámbito en el que va a centrarse este proyecto, que partirá de la base de un autopiloto de código abierto basado en la plataforma Arduino, el Ardupilot[1] o APM (ArduPilotMega) para sobre él desarrollar aplicaciones de toma de datos y comunicación en tiempo real entre uno o más drones y entre un dron con una estación base.

Se dispone además de ordenadores de placa única (también conocidos como SBC o *Single Board Computer*) tipo Raspberry Pi[2], así como módulos de comunicación Wi-Fi.

El proyecto se dividirá en las siguientes fases:

- La fase inicial del proyecto consistirá en el montaje de 3 multirrotores totalmente equipados y listos para poder ser pilotados manualmente por control remoto. Dicho montaje abarcará operaciones como la calibración y el ajuste de la electrónica de control y de potencia, así como la puesta a punto mecánica.
- Una segunda fase, en la que se desarrollará una aplicación software para PC que permita definir una ruta y cargarla en el autopiloto del multirrotor. Esta ruta será posteriormente descrita de forma totalmente autónoma por el multirrotor.
- Una tercera fase que consistirá en lo relativo al desarrollo de un sistema de comunicaciones necesario para comunicar unos multirrotores con otros durante el

[1] http://ardupilot.org/ardupilot/index.html
[2] https://www.raspberrypi.org/

vuelo. Este sistema consistirá en las placas Raspberry Pi y dispositivos inalámbricos de comunicación. El objetivo de esta fase es que al menos dos multirrotores efectúen un vuelo al mismo tiempo, capturando información de algún tipo y que ésta información pueda ser compartida en vuelo entre ellos y relacionada en todo momento con sus coordenadas GPS.

1.2 Objetivos

Este proyecto pretende alcanzar los objetivos que se listan a continuación:

- Puesta en orden de vuelo de una flota de 3 drones de tipología quadcopter.
- Desarrollo de una aplicación para PC que permita definir una ruta mediante una serie de *waypoints* y cargarla en la placa de control del multirrotor.
- Vuelo autónomo de un multirrotor siguiendo la ruta definida en software, sin comunicación con estación base u otros drones.
- Desarrollo de la interfaz de comunicación entre el piloto automático Ardupilot y la placa Raspberry Pi, que permita intercambiar datos de navegación entre ambas.
- Desarrollo de la intercomunicación Wi-Fi entre dos drones, para intercambiar información en tiempo real durante el vuelo de los sensores disponibles en el multirrotor.
- Puesta en vuelo simultánea de varios drones con toma de datos de sensores y comunicación entre ellos.

2 FUNDAMENTOS TEÓRICOS

Solo podemos ver poco del futuro, pero lo suficiente
para darnos cuenta de que hay mucho que hacer.

- Alan Turing -

El presente proyecto se fundamenta sobre dos pilares principales: aeronaves no tripuladas y comunicaciones. Antes de exponer el trabajo práctico llevado a cabo el presente capítulo hace una revisión del estado del arte de las aeronaves no tripuladas, su evolución y sus posibles aplicaciones. Posteriormente se hace una descripción teórica de los multirrotores, aeronave seleccionada para los desarrollos de este proyecto, y por último se realiza una descripción de los distintos sistemas de comunicaciones involucrados.

2.1 Aeronaves no tripuladas (UAVs)

Existe en la actualidad un amplio espectro de aeronaves con capacidad de realizar misiones con un cierto grado de autonomía, lo que conlleva una variedad de términos que son utilizados para referirse a este tipo de aeronaves. En el pasado fueron denominados RPA [1] (Remotely Piloted Aircraft) o UA (Unmanned Aircraft o Uninhabited Aircraft). Estas denominaciones hacen referencia a la ausencia de tripulación en el vehículo, lo que no es necesariamente sinónimo de autonomía.

Figura 1: UAV-500, del fabricante ELIMCO

Se denomina aeronave no tripulada (UAV o Unmanned Aerial Vehicle) a aquella que tiene la capacidad de realizar una determinada misión sin necesidad de tener tripulación a bordo. Esta condición no excluye la posibilidad de que exista un piloto, controlador de misión u otros operadores, quienes pueden realizar su trabajo desde tierra. Por tanto, podemos decir que un RPA siempre es un UAV, mientras que un UAV no necesariamente se tendrá que considerar RPA ya que podría no necesitar de la intervención de un operador externo y ser, por tanto, plenamente autónomo. En la Figura 1: UAV-500, del fabricante ELIMCO se muestra un ejemplo de aeronave no tripulada o UAV.Figura 1: UAV-500, del fabricante ELIMCO

Es conveniente considerar que esta definición podría incluir a algunos artefactos que normalmente quedan excluidos del concepto subyacente de UAV. Así, los globos aerostáticos, como por ejemplo los globos sonda meteorológicos, responden bien a la definición y sin embargo no son considerados UAV al no ser controlables.

La extensión del concepto de vehículo a sistema (UAS o Unmanned Aerial System) refleja el hecho de que muchos sistemas aeronáuticos no tripulados precisan no solo de una aeronave adecuadamente instrumentada sino también de una estación en tierra (GCS o Ground Control Station) que complemente dicha instrumentación así como distintas capacidades embarcadas,

payload (carga de pago) y cuantos equipos de soporte sean necesarios. La Figura 2: UAS Indela Sky, del fabricante ruso Indela. Consta de helicóptero autónomo (UAV) y estación de tierra (GCS) muestra un

ejemplo de UAS comercial, conformado por un helicóptero autónomo y su estación de control en tierra.

Figura 2: UAS Indela Sky, del fabricante ruso Indela. Consta de helicóptero autónomo (UAV) y estación de tierra (GCS)

En los últimos años, la proyección de estos sistemas desde el sector militar al civil ha propiciado que grupos de investigadores, procedentes del área de la robótica, hayan enfocado sus esfuerzos en la investigación y desarrollo de estos vehículos. Por este motivo, se utiliza en ocasiones el término robot aéreo entendiendo como tal a un sistema físico, capaz de desplazarse de manera autónoma o semiautónoma por el aire para realizar diferentes misiones.

Otro término utilizado con frecuencia es dron (*drone* en inglés), el cual se utiliza normalmente para referirse a UAVs de tipo multirrotor y que no precisa si el guiado del mismo es automático o manual.

2.1.1 Clasificación de los sistemas UAS

Debido a la diversidad de tipologías, tamaños, capacidades y utilidades de los UAS actuales, es complicado establecer una clasificación única. La Tabla 1: clasificación UAS en función de alcance, altitud, autonomía y MTOW.[2] propone una clasificación general, teniendo en cuenta los siguientes parámetros:

- Alcance
- Altitud
- Autonomía de vuelo
- Peso máximo al despegue (MTOW).

La Figura 3: Clasificación UAS muestra las distintas tipologías en función de la altitud de vuelo y el alcance.

Categoría	Alcance (km)	Altitud (m)	Autonomía (h)	MTOW (kg)
Estratosféricos	>2000	20000-30000	48	<3000
Elevada altitud y gran autonomía (HALE)	>2000	20000	48	15000
Media altitud y gran autonomía (MALE)	>500	14000	24-48	1500
Baja altitud y gran autonomía (LALE)	>500	3000	Aprox. 24	Aprox. 30
Baja altitud y amplia penetración (LADP)	>250	50-9000	0,25 – 1	350
Medio alcance	70 - >500	8000	6-18	1250
Corto alcance	10-70	3000	3-6	200
Mini	<10	<300	<2	<30
Micro	<10	<250		<1

Tabla 1: clasificación UAS en función de alcance, altitud, autonomía y MTOW.

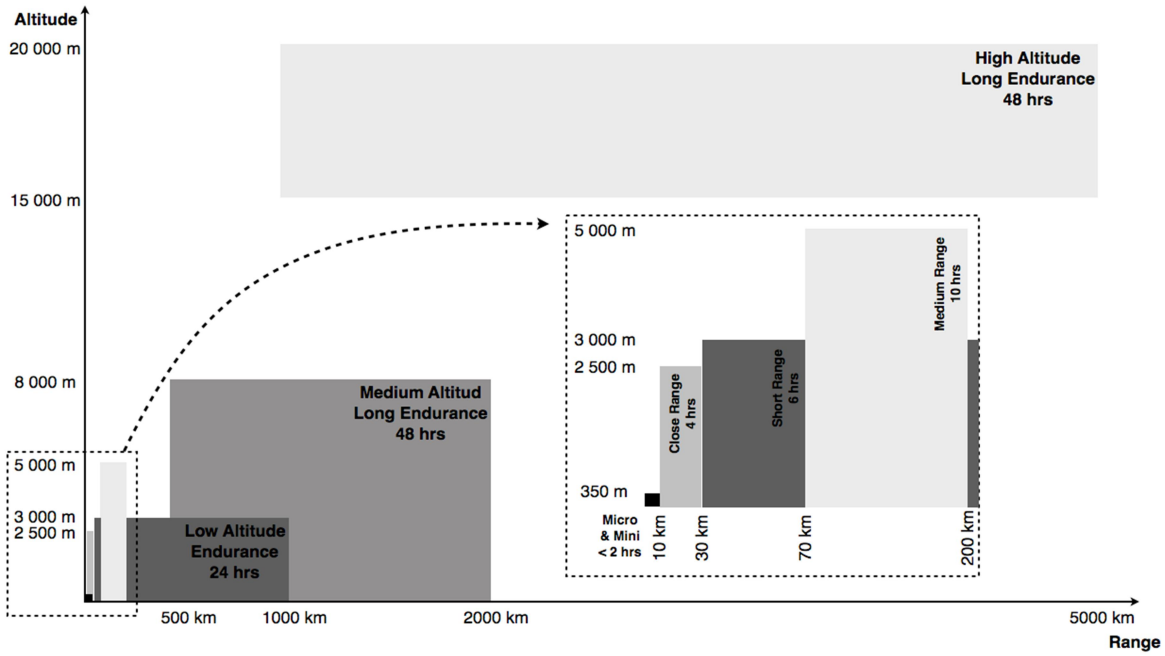

Figura 3: Clasificación UAS

Atendiendo a criterios similares, otra posible clasificación más detallada de los UAS podría ser la propuesta en la Tabla 2: clasificación detallada UAS. (A = Avión, H = Helicóptero, o = otros):

Categoría	Acrónimo	Alcance (km)	Techo (m)	Autonomía (horas)	MTOW (kg)	Tipo de aeronave
Micro	μ(Micro)	<10	250	1	<5	H, A, o
Mini	Mini	<10	150 a 300	<2	<30	H, A, o
Alcance cercano	CR	10 a 30	3.000	2 a 4	150	H, A, o
Alcance corto	SR	30 a 70	3.000	3 a 6	200	A, o
Alcance medio	MR	70 a 200	5.000	6 a 10	1.250	A, o
Altitud baja Penetración profunda	LADP	>250	50 a 9000	0,5 a 1	350	A
Autonomía media	MRE	>500	8000	10 a 18	1250	A, H
Autonomía alta, altitud baja	LALE	>500	3000	>24	<30	A
Autonomía alta altitud media	MALE	>500	14000	24 a 48	1.500	A, H
Autonomía alta altitud alta	HALE	>2000	20000	24 a 48	12.000	A

Combate	UCAV	Aprox. 1500	10000	aprox. 2	10.000	H, A
Ofensivo	LETH	300	4000	3 a 4	250	A
Señuelo	DEC	0 a 500	5000	<4	250	
Estratosférico	STRATO	>2000	Entre 20000 y 30000	>48	N/A	A
Exo-estratosférico	EXO	N/A	> 30000	N/A	N/A	A

Tabla 2: clasificación detallada UAS.

En cuanto al tipo de aeronave, puede efectuarse una clasificación en función de la forma de despegar, distinguiendo así entre aeronaves de despegue vertical y aeronaves de despegue no vertical. La Figura 4: clasificación UAS según forma de despegue muestra dicha clasificación:

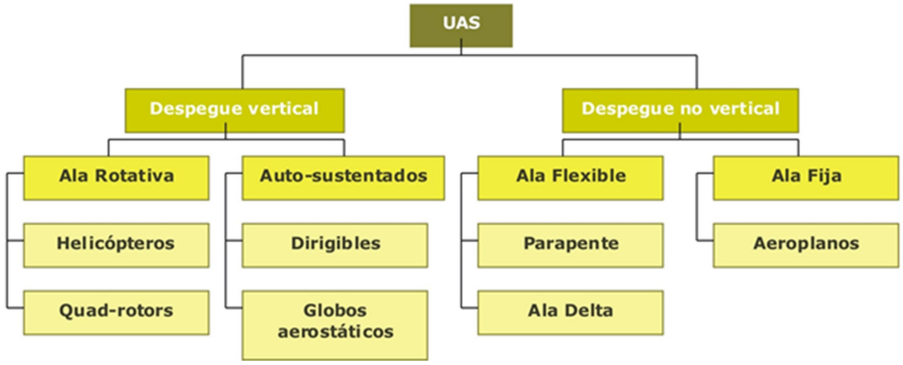

Figura 4: clasificación UAS según forma de despegue

La Tabla 3: características de las distintas tipologías de UAS muestra las capacidades de las principales tipologías de aeronaves que usualmente son empleadas como UAS:

Característica	Helicóptero	Aeroplano	Dirigible	Multirrotor
Capacidad de vuelo estacionario	***		****	***
Velocidad de desplazamiento	***	****	*	**
Maniobrabilidad	***	*	*	****
Autonomía de vuelo	**	***	****	*
Resistencia a perturbaciones externas (viento)	**	****	*	**

Auto estabilidad	*	***	****	**
Capacidad de vuelos verticales	****	*	**	****
Capacidad de carga	***	****	*	**
Capacidad de vuelo en interiores	**	*	***	****
Techo de vuelo	**	****	***	*

Tabla 3: características de las distintas tipologías de UAS

Los sistemas UAS pueden, con independencia de los criterios anteriores, ser clasificados también en función de su nivel de autonomía, distinguiéndose las siguientes categorías:

- *Autónomo y adaptativo*: el UAV está totalmente gobernado por sus sistemas de abordo, sin intervención de un operador en tierra. Tiene capacidad para re-planificar su vuelo, puede interacturar con otros sistemas y tomar decisiones por sí mismo.
- *Monitorizado*: opera de forma autónoma y un operador controla la retroalimentación del UAV. El operador no controla directamente los mandos de la aeronave, pero puede tomar decisiones por él.
- *Supervisado*: el UAV realiza algunas operaciones de forma autónoma. El control recae en su gran mayoría sobre el operador.
- *Autónomo no adaptativo (pre-programado)*: el UAV obedece a una rutina pre-programada y no tiene la capacidad de cambiar esa rutina para adaptarla a los cambios externos.
- *Mando directo por un operador (R/C)*: el UAV responde directamente a los mandos de un operador.

2.1.2 Aplicaciones de los sistemas UAV

Las aeoronaves no tripuladas son una tecnología en pleno proceso de expansión, con una gran variedad de aplicaciones potenciales.

Se puede distinguir entre aplicaciones militares, de defensa y seguridad, y apliaciones civiles [3]. Si bien a lo largo del siglo XX la operación de aeronaves no tripuladas autónomas (sin tener en cuenta aeromodelos radiocontrolados) rara vez se realizaba en un ámbito no militar, actualmente cada vez son más las aplicaciones civiles de las mismas. A continuación se describen brevemente

algunas de estas aplicaciones.

2.1.2.1 Aplicaciones militares

Entre las aplicaciones de índole militar podemos destacar las siguientes:

- Misiones de inteligencia, vigilancia y reconocimiento.
- Apoyo a Instituciones del Estado con competencias en la lucha contra actividades ilegales tales como la inmigración ilegal, narcotráfico, tráfico de armas, etc.
- Sistemas de combate aéreo, UCAVs (*Unmanned Combat Aircraft Vehicle*)
- Apoyo a misiones humanitarias.
- Vigilancia de infraestructuras críticas (aeropuertos, telecomunicaciones, líneas eléctricas, oleoductos, etc) así apoyo a instituciones del estado.
- Avión blanco, para pruebas de armamento.
- Salvamento y vigilancia marítima.

La Figura 5: UAV ALO (Avión Ligero de Observación) muestra el ALO (Avión Ligero de Observación), perteneciente al Instituto Nacional de Técnica Aeroespacial (INTA). Se trata de una plataforma UAV de corto y medio alcance, capaz de operar de forma completamente autónoma y equipada con cámaras en espectro visible e IR.

Figura 5: UAV ALO (Avión Ligero de Observación)

2.1.2.2 Aplicaciones civiles

Algunas de las aplicaciones de sistemas UAV en un entorno civil son las siguientes [3]:

- Gestión de emergencias y de desastres naturales (incendios forestales, terremotos, etc).
- Monitorización medioambiental, meteorología y aplicaciones científicas.
- Monitorización de la flora y la fauna para usos científicos en oceanografía, geofísica, actividad sísmica, y en general para la observación e investigación climatológica y meteorológica.
- Inspección de instalaciones, infraestructuras y edificios empleando sensores electro-ópticos y radiométricos.
- Agricultura, aplicaciones forestales y pesqueras, como la generación de inventarios de áreas de cultivos o forestales, la vigilancia de cosechas y fumigación.
- Usos cartográficos y catastrales, para construir mapas del terreno con mayor precisión que los obtenidos con aeronaves convencionales.
- Fotografía aérea y cinematografía.
- Telecomunicaciones. Los UAS pueden aplicarse como plataformas de apoyo a las comunicaciones.

- Vigilancia de tráfico, como la monitorización de tráfico en carreteras, puertos, costas y vías fluviales.
- Divertimento del usuario, a nivel de aficionado.

2.2 Multirrotores

De entre los distintos tipos de UAV, un helicóptero autónomo posee ciertas propiedades, como son su inherente habilidad de volar a baja velocidad, estacionariamente, lateralmente e incluso realizar maniobras en espacios reducidos, que lo convierten en el vehículo aéreo perfecto para casi cualquier tarea dentro de un rango de acción limitado. La capacidad de despegue vertical y vuelo estacionario supone también un ahorro considerable en los costes de operación, ya que elimina la necesidad de disponer de un campo de vuelo para poder actuar. Este es el principal motivo por el que las aeronaves elegidas para este proyecto han sido de este tipo.

El multirrotor es un tipo de helicóptero el cual es accionado mediante un cierto número de rotores [1]. De entre las distintas tipologías de multirrotores la más extendida es el llamado quadcopter o quadrotor (Figura 6: quadrotor SOLO, fabricado por 3DRobotics), que dispone de 4 rotores coplanarios, situados en posición horizontal, de forma que expulsan la corriente de aire hacia abajo. Dichos rotores están colocados formando un cuadrado geométrico, donde el centro de masas del quadrotor se encuentra en el centro de dicho cuadrado.

Los quadrotors han experimentado un auge reciente debido a la evolución del hardware disponible. Así pues, los recientes avances tecnológicos en actuadores y sensores en escala reducida (MEMS- Micro ElectroMechanical Systems), así como en almacenamiento de energía y procesamiento de datos han sido cruciales para que hoy en día pueda disponerse de un quadrotor a un precio no demasiado elevado (menos de 1000 euros).

Figura 6: quadrotor SOLO, fabricado por 3DRobotics

El control de los movimientos de un quadrotor se lleva a cabo ajustando las velocidades angulares de los distintos rotores, los cuales están accionados por motores eléctricos (un motor en cada rotor). Para lograr movimiento hacia adelante la velocidad de los rotores traseros debe ser aumentada y, simultáneamente, la velocidad de los delanteros debe ser disminuida.

El desplazamiento lateral se ejecuta con un procedimiento análogo, acelerando los rotores izquierdos y frenando los derechos para realizar un desplazamiento lateral hacia la derecha. El movimiento de guiñada o *yaw*, es decir, realizar un giro alrededor de un eje vertical, se obtiene a partir de la diferencia en el par de torsión entre cada par de rotores, es decir, se aceleran los dos rotores que giran en sentido horario mientras se desaceleran los rotores que giran en sentido anti-horario, y vice-versa.

Siguiendo unos criterios más formales, un helicóptero de tipo quadrotor es un sistema mecánico subactuado, con 4 entradas de control (que corresponden con los 4 rotores) y 6 grados de libertad (3 grados de libertad de orientación espacial y 3 grados de libertad de posición). A bajo nivel, se dispone del control en velocidad de cada uno de los motores, los cuales tienen como referencia consignas de velocidad proporcionadas por un controlador externo. Desde el lazo externo se asume que la dinámica del control en velocidad para alcanzar la velocidad de referencia de cada uno de los motores es despreciable.

Para aumentar tanto la fiabilidad como las prestaciones de estos sistemas, se suele recurrir a sistemas de control avanzados que permitan tener en cuenta, por una parte, la complejidad de estos sistemas, y por otra, las incertidumbres propias de cualquier modelado. Los objetivos de un sistema de control de vuelo, en función de la autonomía que alcance el sistema, pueden clasificarse en tres fases [4]:

- Sistema para incrementar la estabilidad (*Stability Augmentation Systems*):

  Este tipo de sistemas persigue ayudar al pilotaje del vehículo, estabilizando el sistema con un control de bajo nivel. Así se evita que el piloto deba actuar en base al comportamiento dinámico de un sistema, que una vez alejado de cierto punto de equilibrio, deja de ser intuitivo para el razonamiento humano.

- Sistemas para incrementar el comportamiento (*Control Augmentation Systems*): Estos sistemas están en un nivel jerárquico superior a los SAS. Así, además de estabilizar al vehículo, estos sistemas deben ser capaces de proporcionar una respuesta con ciertas prestaciones a referencias que proporcione el piloto, como por ejemplo, el seguimiento del ángulo de cabeceo.
- Sistemas de pilotaje automático (Autopilotos): Constituyen el nivel de control jerárquicamente superior. Son sistemas de control totalmente automáticos que son capaces de realizar por si solos ciertos tipos de maniobras, como por ejemplo, el despegue, el aterrizaje, o vuelo estacionario a cierta altura.

Los quadcopter han recibido mucha atención por parte de investigadores, ya sea para el estudio de su control y guiado como por la infinidad de aplicaciones posibles. Sin duda, otro punto a favor del multirrotor frente a los helicópteros tradicionales es su simpleza mecánica. La Figura 7: rotor principal de un helicóptero radiocontrolado del fabricante ALIGN. muestra el rotor principal de un helicótero radiocontrolado tradicional. Como puede apreciarse, los helicópteros tradicionales tienen elementos mecánicos muy complejos que requieren de un mantenimiento y una puesta a punto más extendidos que en un multirrotor.

Figura 7: rotor principal de un helicóptero radiocontrolado del fabricante ALIGN.

2.2.1 Modelo cinemático del quadcopter

En la Figura 8: sistema de referencia en ejes cuerpo (x_B, y_B, z_B),

velocidades angulares (w_1, w_2, w_3, w_4), fuerzas propulsoras, (f_1, f_2, f_3, f_4)

y pares de los motores $(T_{M1,}T_{M2}, T_{M3}, T_{M4})$.

Ángulos de Euler (izqda) [5] se presenta la estructura de un quadcopter así como el sistema de referencia en ejes cuerpo, x_B, y_B, z_B, las fuerzas correspondiente al empuje de cada motor, f_1, f_2, f_3, f_4 y los momentos inducidos por el giro de los motores, $T_{M1,}T_{M2}, T_{M3}, T_{M4}$.

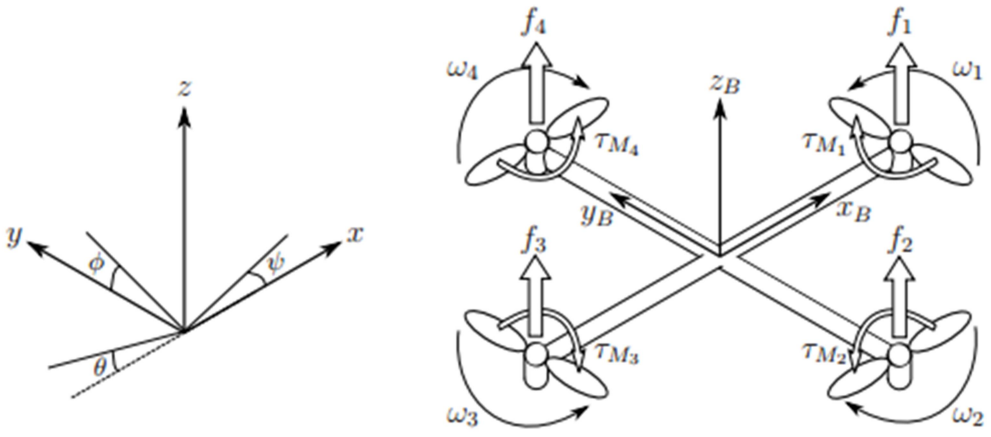

Figura 8: sistema de referencia en ejes cuerpo (x_B, y_B, z_B),
velocidades angulares (w_1, w_2, w_3, w_4), fuerzas propulsoras, (f_1, f_2, f_3, f_4)
y pares de los motores $(T_{M1,}T_{M2}, T_{M3}, T_{M4})$.
Ángulos de Euler (izqda)

La posición absoluta del quadrotor se define respecto del sistema de referencia inercial definido por los ejes *x, y, z.*

$$\xi = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

La actitud del quadrotor, es decir, su inclinación o posición angular se define en el sistema de referencia inercial a través de los ángulos de Euler, η. El ángulo de cabeceo (Pitch, θ) determina la rotación del quadrotor alrededor del eje y. El ángulo de alabeo (Roll, φ) determina la rotación alrededor del eje x y el ángulo de guiñada (Yaw, ψ) determina la rotación alrededor del eje z.

$$\eta = \begin{pmatrix} \varphi \\ \theta \\ \psi \end{pmatrix}$$

El vector q contiene los vectores de posición lineal y de posición angular.

$$q = \begin{pmatrix} \xi \\ \eta \end{pmatrix}$$

El origen del sistema de referencia en ejes cuerpo es el centro de masas del quadcopter. El vector V_B contiene las velocidades lineales, así como el vector ν contiene las velocidades angualares.

$$V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}$$

$$\nu = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

La matriz de rotación para pasar del sistema de referencia en ejes cuerpo al sistema de referencia inercial es la siguiente:

$$R = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\varphi - S_\psi C_\varphi & C_\psi S_\theta C_\varphi + S_\psi S_\varphi \\ S_\psi C_\theta & S_\psi S_\theta S_\varphi + C_\psi C_\varphi & S_\psi S_\theta C_\varphi - C_\psi S_\varphi \\ -S_\theta & C_\theta S_\varphi & C_\theta C_\varphi \end{bmatrix}$$

La matriz R es ortogonal, siendo por tanto su inversa igual a su traspuesta. La matriz

traspuesta de R es la matriz de rotación del sistema de referencia inercial al sistema de referencia en ejes cuerpo.

Las matrices para transformar las velocidades angulares del sistema de referencial inercial al sistema en ejes cuerpo, y del sistema en ejes cuerpo al sistema inercial son W_η y ${W_\eta}^{-1}$ respectivamente:

$$\dot{\eta} = {W_\eta}^{-1}\nu$$

$$\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & S_\varphi T_\theta & C_\theta T_\theta\\ 0 & C_\varphi & -S_\varphi\\ 0 & S_\varphi/C_\theta & C_\varphi/C_\theta\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix}$$

$$\nu = W_\eta\dot{\eta}$$

$$\begin{bmatrix}p\\ q\\ r\end{bmatrix} = \begin{bmatrix}1 & 0 & -S_\theta\\ 0 & C_\varphi & C_\theta S_\varphi\\ 0 & -S_\varphi & C_\theta C_\varphi\end{bmatrix}\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}$$

En la tipología más habitual, el chasis del quadrotor tiene una estructura doblemente simétrica, en la cual los brazos están alineados con los ejes cuerpo x e y. En este caso, el tensor de inercia es una matriz diagonal, y la inercia respecto del eje x es igual a la inercia respecto del eje y.

$$I = \begin{bmatrix}I_{xx} & 0 & 0\\ 0 & I_{yy} & 0\\ 0 & 0 & I_{zz}\end{bmatrix}$$

La velocidad angular del rotor i, w_i, crea una fuerza f_i en la dirección del eje de dicho rotor. La velocidad y aceleración angular de dicho rotor también genera un par $\tau_{M,i}$:

$$f_i = k\, {w_i}^2$$

$$\tau_{M,i} = b\, {w_i}^2 + I_M\dot{w}_i$$

El parámetro k es el coeficiente de sustentación, b es el coeficiente de arrastre y el momento de inercia del rotor es I_M. Usualmente el efecto de $\dot{w}_i$ se omite por ser relativamente pequeño.

Las fuerzas producidas por los 4 rotores crean un empuje T en la dirección del eje cuerpo z. El vector de pares τ_B contiene los pares. El vector de pares, τ_B, contiene los pares en las direcciones de

los ángulos en ejes cuerpo, $\tau_{\varphi}, \tau_{\theta}, \tau_{\psi}$.

$$T = \sum_{i=1}^{4} f_i = k \sum_{i=1}^{4} {w_i}^2$$

$$T^B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

$$\tau_B = \begin{bmatrix} \tau_{\varphi} \\ \tau_{\theta} \\ \tau_{\psi} \end{bmatrix} = \begin{bmatrix} lk(-{w_2}^2 + {w_4}^2) \\ lk(-{w_1}^2 + {w_3}^2) \\ \sum_{i=1}^{4} \tau_{M,i} \end{bmatrix}$$

El parámetro l es la distancia entre cada rotor y el centro de masas del quadcopter. En la última ecuación podemos ver cómo el giro en roll se obtiene frenando el segundo rotor y acelerando el cuarto. De igual modo, el movimiento en pitch se obtiene decelerando el rotor 1 y acelerando el rotor 3. El movimiento en yaw se obtiene acelerando dos rotores opuestos y decelerando los otros dos.

2.2.2 Dinámica del quadcopter

Si se acepta que el chasis del quadrotor es un sólido rígido [6] podemos utilizar las ecuaciones de Newton-Euler para describir su dinámica. En el sistema de referencia en ejes cuerpo, la fuerza sobre el centro de masas, $m\dot{V}_B$, y la fuerza centrifuga, $\nu \times (mV_B)$, son iguales a la fuerza gravitatoria, $R^T G$ y al empuje total de los rotores, T_B, respectivamente.

$$m\dot{V}_B + \nu \times (mV_B) = R^T G + T_B$$

En el sistema de referencia inercial, por definición, no existen fuerzas centrífugas. Únicamente la fuerza gravitatoria y el empuje de los rotores contribuyen a la aceleración del quadrotor.

$$m\ddot{\xi} = G + RT_B$$

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -g\begin{bmatrix}0\\0\\1\end{bmatrix} + \frac{T}{m}\begin{bmatrix}C_\psi S_\theta C_\varphi + S_\psi S_\varphi\\ S_\psi S_\theta C_\varphi - C_\psi S_\varphi\\ C_\theta C_\varphi\end{bmatrix}$$

En el sistema de referencia en ejes cuerpo, la aceleración angular $I\dot{\nu}$, las fuerzas centrípetas $\nu \times (I\nu)$ y las fuerzas giroscópicas Γ deben ser iguales al par externo total, τ.

$$I\dot{\nu} + \nu \times (I\nu) + \Gamma = \tau$$

$$\dot{\nu} = I^{-1}\left(-\begin{bmatrix}p\\q\\r\end{bmatrix} \times \begin{bmatrix}I_{xx}p\\I_{yy}q\\I_{zz}r\end{bmatrix} - I_r\begin{bmatrix}p\\q\\r\end{bmatrix} \times \begin{bmatrix}0\\0\\1\end{bmatrix} w_\Gamma + \tau\right)$$

$$\begin{bmatrix}\dot{p}\\\dot{q}\\\dot{r}\end{bmatrix} = \begin{bmatrix}(I_{yy} - I_{zz})qr/I_{xx}\\(I_{zz} - I_{xx})pr/I_{yy}\\(I_{xx} - I_{yy})pq/I_{zz}\end{bmatrix} - I_r\begin{bmatrix}q/I_{xx}\\-p/I_{yy}\\0\end{bmatrix} w_\Gamma + \begin{bmatrix}\tau_\varphi/I_{xx}\\\tau_\theta/I_{yy}\\\tau_\psi/I_{zz}\end{bmatrix}$$

$$w_\Gamma = w_1 - w_2 + w_3 - w_4$$

Las aceleraciones angulares en el sistema inercial se obtienen a partir de las aceleraciones en el sistema ejes cuerpo con la matriz $\mathrm{W_\eta}^{-1}$ y su derivada.

$$\ddot{\eta} = \frac{\mathrm{d}}{\mathrm{dt}}\left(\mathrm{W_\eta}^{-1}\nu\right) = \frac{\mathrm{d}}{\mathrm{dt}}\left(\mathrm{W_\eta}^{-1}\right)\nu + \mathrm{W_\eta}^{-1}\dot{\nu}$$

$$\ddot{\eta} = \begin{bmatrix}0 & \dot{\varphi}C_\varphi T_\theta + \dot{\theta}S_\varphi/{C_\theta}^2 & -\dot{\varphi}S_\varphi C_\theta + \dot{\theta}C_\varphi/{C_\theta}^2\\ 0 & -\dot{\varphi}S_\varphi & -\dot{\varphi}C_\varphi\\ 0 & \frac{\dot{\varphi}C_\varphi}{C_\theta} + \dot{\varphi}S_\varphi T_\theta/C_\theta & -\frac{\dot{\varphi}S_\varphi}{C_\theta} + \dot{\theta}C_\varphi T_\theta/C_\theta\end{bmatrix}\nu + \mathrm{W_\eta}^{-1}\dot{\nu}$$

2.2.3 Efectos aerodinámicos

Al modelo dinámico anterior se puede incluir un término que represente el arrastre aerodinámico generado por la resistencia aerodinámica.

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -g\begin{bmatrix}0\\0\\1\end{bmatrix} + \frac{T}{m}\begin{bmatrix}C_\psi S_\theta C_\varphi + S_\psi S_\varphi\\ S_\psi S_\theta C_\varphi - C_\psi S_\varphi\\ C_\theta C_\varphi\end{bmatrix} - \frac{1}{m}\begin{bmatrix}A_x & 0 & 0\\0 & A_y & 0\\0 & 0 & A_Z\end{bmatrix}\begin{bmatrix}\dot{x}\\\dot{y}\\\dot{z}\end{bmatrix}$$

A_x, A_y y A_Z representan los coeficientes de arrastre en las distintas direcciones del sistema de referencia inercial.

2.2.4 Control

El sistema de control del quadcopter puede ser descompuesto en dos subsistemas de control: el subsistema rotacional y el subsistema traslacional. En los casos en que el vehículo está siendo operado por un piloto, éste realiza a través de un mando remoto una serie de entradas al subsistema rotacional del sistema, el cual se encarga de estabilizar el quadcopter. Sin embargo, el subsistema traslacional está siendo totalmente interiorizado por el piloto y es él quien implementa tanto el control de posición como la planificación de la trayectoria.

Sin embargo, cuando el guiado del multirrotor esté siendo realizado de forma automática (como por ejemplo en los modos *Auto*, *Guided* o *Loiter* del autopiloto APM), la controladora asume también el control del subsistema traslacional, teniendo que actuar sobre el control de posición y la planificación de la trayectoria. A continuación se explicará cómo implementa el autopiloto APM ambos subsistemas de control.

2.2.4.1 Control del subsistema rotacional

En lo que respecta al subsistema rotacional, podemos distinguir los siguientes elementos:

- Entradas: en función del modo de vuelo las entradas a este sistema serán o los ángulos de inclinación deseados o bien las velocidades de rotación en cada uno de los ejes.
- Retroalimentación: en el caso del subsistema rotacional, el elemento encargado de cerrar el bucle de control es la Unidad de Medición Inercial (IMU), que obtiene información tanto de los ángulos de inclinación (en base a lecturas de acelerómetros) como de las velocidades angulares (en base a las lecturas de los giróscopos).

APM distingue dos tipos de controladores en este subsistema: los *body frame controllers* y los *earth frame controllers* (Figura 9: bucles de control del autopiloto APM) [7].

- *Earth frame controllers:* en una iteración del bucle de control del subsistema rotacional, éstos actúan en primer lugar. Son los encargados de, en función del ángulo de inclinación deseado (sticks) y la inclinación real (lectura de los acelerómetros), establecer una velocidad de rotación deseada para minimizar el error y ajustar ambos valores. Es un controlador tipo proporcional.
- *Body frame controllers:* actúan a continuación de los *earth frame controllers*, como puede verse en el esquema. Toman como entrada la velocidad de rotación deseada en cada eje (información proveniente de los earth frame controllers) y la lectura de velocidad de

rotación real (gyros), e implementan controladores tipo Proporcional-Integral-Derivativo (PID) para ajustar la velocidad deseada a la velocidad real.

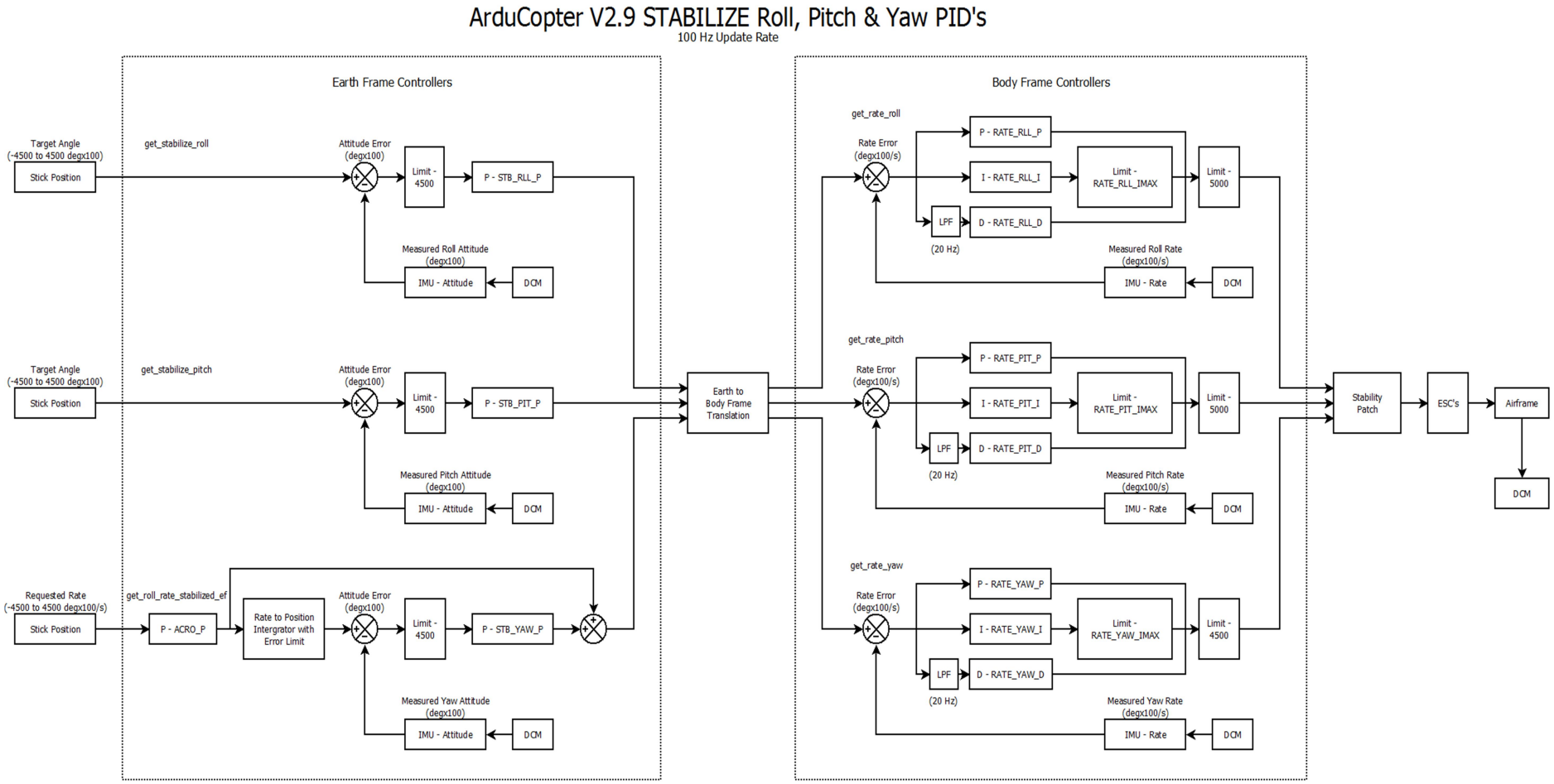

Figura 9: bucles de control del autopiloto APM

2.2.4.2 Control del subsistema traslacional

En algunos modos de vuelo, como lo son los modos Loiter, Guided o Mission, la placa implementa el control del subsistema traslacional además del control del subsistema rotacional. Existen varias posibilidades en cuanto a sensores que cierran los bucles de control en el subsistema traslacional, por ejemplo los siguientes:

- Señal GPS
- Sensores barométricos
- Sonar
- Control visual

En estos casos, el sistema de control del subsistema rotacional se mantiene igual que en pilotaje manual, cambiando únicamente la entrada a este sistema. Anteriormente, la entrada al lazo de control rotacional era la posición de los sticks del piloto que, en función del modo de vuelo, proporcionaba una información o bien referente a la velocidad angular de cada uno de los ejes o bien referente a la inclinación deseada en cada eje.

El subsistema traslacional se encargará de calcular, en función de los valores de longitud, latitud y altura deseados (longitud, latitud y altura del *waypoint* siguiente) y de la longitud, latitud y altura reales del quadcopter, los angulos de cabeceo, alabeo y guiñada adecuados para conseguirlos. Estos valores actuarán a su vez como inputs del subsistema rotacional, que será el encargado de conseguir dichos ángulos.

2.3 Comunicaciones en UAVs

En función de la tipología, modo de operación y aplicación de un sistema UAV, pueden ser necesarios diversos sistemas de comunicaciones entre el mismo y la estación de control en tierra, o GCS. A continuación se describen los sistemas principales, si bien, dado el gran abanico de posibilidades que ofrecen los sistemas UAV, la clasificación es casi infinita.

2.3.1 Radioenlace de control manual

Se trata de un enlace de radio que permite al operador pilotar manualmente desde tierra el sistema UAV. Es el único sistema de comunicaciones existente en los UAV más simples, los cuales tienen que ser siempre controlados por un piloto desde tierra.

Figura 10: emisora de radio Futaba T14SG

Tradicionalmente se han utilizado sistemas de radiocontrol de aeromodelismo para este cometido, si bien actualmente existen equipos más avanzados y con prestaciones mucho más elevadas. Dichos sistemas constan de un equipo embarcado (el receptor, Figura 11: receptor de radio Futaba R6014FS, 2.4Ghz) y un equipo en tierra (la emisora, Figura 10: emisora de radio Futaba T14SG).

Figura 11: receptor de radio Futaba R6014FS, 2.4Ghz

Las frecuencias más usuales en que funcionan estos equipos son 35Mhz y 2.4Ghz. En la mayoría de los casos, la emisora se divide en dos subsistemas que son físicamente separables:

- El sistema modulador: en función de la posición de las palancas de mando o *sticks* y de los interruptores, el modulador crea una señal del tipo PPM (*Pulse Position Modulation* o modulación por posición de pulso) en la que se multiplexa toda esa información.

- El sistema emisor: es el encargado de transmitir la señal generada por el modulador hacia el receptor.

En muchas emisoras, el sistema emisor puede sustituirse por otro, lo cual puede ser útil para transmitir en una frecuencia distinta o para disponer de mayor potencia de emisión.

Respecto del control que el piloto ejerce sobre el UAV, existen dos posibilidades alternativas las cuales se explican a continuación:

- La aeronave no tiene ningún tipo de sistema de control a bordo, recayendo toda la responsabilidad del control de la misma sobre el piloto. En este caso, el receptor de radio a bordo del UAV está directamente conectado a los distintos actuadores, como pueden ser motores o superficies de control.
- La aeronave tiene algún tipo de sistema autopiloto que ayuda a la estabilización de la misma. Es el caso de los multirrotores, que son sistemas inestables y necesitan de un sistema estabilizador para mantener un vuelo controlado. En este caso el receptor de radio está conectado al sistema autopiloto el cual a su vez se comunica con los actuadores, que en el caso de un multirrotor son sus motores eléctricos. De este modo, el piloto controla el UAV en tiempo real a través de un sistema de radio, pero sus órdenes son interpretadas por el autopiloto y es éste el encargado de controlar los actuadores.

2.3.1.1 Ampliación del alcance de control

El sistema de comunicaciones de control manual puede ser tanto el sistema principal de control de un UAV, en el caso de un UAV radiocontrolado, como un sistema de respaldo en un UAV que permita operación autónoma. En este caso, el sistema de radiocontrol puede utilizarse para las operaciones de despegue/aterrizaje, si es que estos no pueden ser efectuados de forma autónoma. Igualmente, ante un fallo del control autónomo, el sistema de radiocontrol podría utilizarse para recuperar el UAV manualmente.

Sin embargo, estas operaciones anteriormente descritas, están limitadas en el espacio al alcance visual del piloto. Ante este problema, la solución son los sistemas FPV, que se describen en el siguiente apartado.

2.3.2 Radioenlace de video: Sistema FPV

El radioenlace de video es un sistema que permite al operador disponer en tierra de una imagen en tiempo real proveniente de una cámara a bordo del UAV. Si la cámara está correctamente orientada, el piloto podría pilotar el UAV con el radioenlace manual como si estuviera a bordo del mismo. A este tipo de operación se le conoce como vuelo en primera persona o FPV (*First Person View).*

Figura 12: conjunto transmisor y receptor de vídeo para vuelo FPV, fabricado por Boscam. Banda de 5.8Ghz.

Existen radioenlaces de vídeo en diversas bandas de frecuencias, siendos los más utilizados en 1.2Ghz, 2.4Ghz y 5.8GHz (Figura 12: conjunto transmisor y receptor de vídeo para vuelo FPV, fabricado por Boscam.). En función de la potencia de emisión y de la ganancia de las antenas utilizadas, el radio de acción del sistema FPV puede variar de unos cientos de metros a decenas de kilómetros. Para aumentar aún más el alcance, se utilizan antenas direccionales en el equipo receptor, instaladas sobre plataformas robotizadas que las orientan en la dirección en que se encuentra el UAV, de modo que la recepción sea óptima.

Los sistemas OSD (*On Screen Display*) permiten sobreimpresionar información del vuelo en el vídeo, convirtiendo el radioenlace de vídeo también en un enlace de telemetría.

En ocasiones el objetivo del radioenlace de vídeo no es pilotar el UAV sino tener una buena imagen de los terrenos sobrevolados por el UAV. En estas ocasiones la cámara suele montarse sobre un dispositivo llamado gimbal (Figura 13: gimbal de un quadrotor DJI Inspire 1.), el cual estabiliza la cámara y la mantiene fija en la orientación deseada por el operador.

Figura 13: gimbal de un quadrotor DJI Inspire 1.

2.3.3 Radioenlace de datos

El radioenlace de datos es un sistema de comunicaciones bidireccional el cual tiene las siguientes aplicaciones:

- Telemetría: el operador recibe a través de dicho radioenlace información del funcionamiento del UAV y de los distintos sensores a bordo. Dicha información puede incluir:
    - Posición GPS
    - Velocidad respecto del suelo
    - Velocidad aerodinámica
    - Altura
    - Combustible/carga restante
- Comando: el operador puede enviar comandos al autopiloto a través de este radioenlace. Algunos posibles comandos son:
    - Indicar una waypoint al que el UAV debe dirigirse
    - Orden de despegue/aterrizaje
    - Cambio de altura
    - Cambio de velocidad
    - Requerimiento de algún dato que no sea enviado por defecto con los datos de telemetría.

El autopiloto utilizado en el presente proyecto, el APM 2.5, incluye un sistema de radioenlace de datos en la frecuencia de 915MHz (Figura 14: módulos de telemetría 3DRobotics. 915MHz.), si bien también lo hay disponible en 468MHz. Consiste en dos módems idénticos, uno de los cuales va embarcado en el UAV y

conectado al autopiloto, y el otro se encuentra en tierra conectado a un PC, dispositivo iOS o Android.

Figura 14: módulos de telemetría 3DRobotics. 915MHz.

Dichos módems funcionan como un puerto serie UART inalámbrico y tienen una potencia de 100mW. Existen alternativas de mucha más potencia y, por tanto, más alcance, para sistemas UAV.

2.3.4 Radioenlace secundario de datos

En algunos casos puede resultar práctico disponer de un segundo sistema de radioenlace de datos, que pueda extender el alcance del sistema primario y servir de respaldo en caso de fallo del mismo. Algunas posibilidades son las siguientes:

- Radioenlace satelital: conexión entre el UAV y la GCS por satélite. Tiene la ventaja de ser un sistema que ofrece, en función de la red satelital elegida, cobertura global. Como desventajas podemos citar el coste de utilizar dichas redes y que la tasa de transferencia de datos no es muy alta. La Figura 15: transceptor satelital Iridium 9602 muestra un transceptor Iridium.

Figura 15: transceptor satelital Iridium 9602

- Radioenlace GSM: radioenlace entre UAV y GCS utilizando la red de telefonía móvil. Permite disponer de cobertura en todas las zonas del mundo que tengan cobertura móvil, y, en función del tipo de red existente, podría permitir incluso el envío de imágenes. Como contrapartida, habría que costear la tarifa de datos móviles correspondiente.

2.3.5 Sistemas de comunicaciones extra

Algunos sistemas UAV tienen finalidades relacionadas con ofrecer servicios de comunicaciones, siendo necesarios por tanto de dispositivos de comunicaciones específicos.

Como ejemplo de este concepto, el proyecto UAVNet [8] consiste en una arquitectura de red inalámbrica aérea de establecimiento automático. Los nodos aéreos, UAVs de tipo quadcopter, se inerconectan entre sí formando una red inalámbrica 802.11s. La red implementada tiene la capacidad de conectar entre sí dos sistemas finales (generalmente ubicados en tierra) mediante la creación de una cadena de nodos, formado por uno o varios multirrotores (Figura 16: interconexión entre dos sistemas finales mediante

dos nodos aéreos en el seno del proyecto UAVNet.).

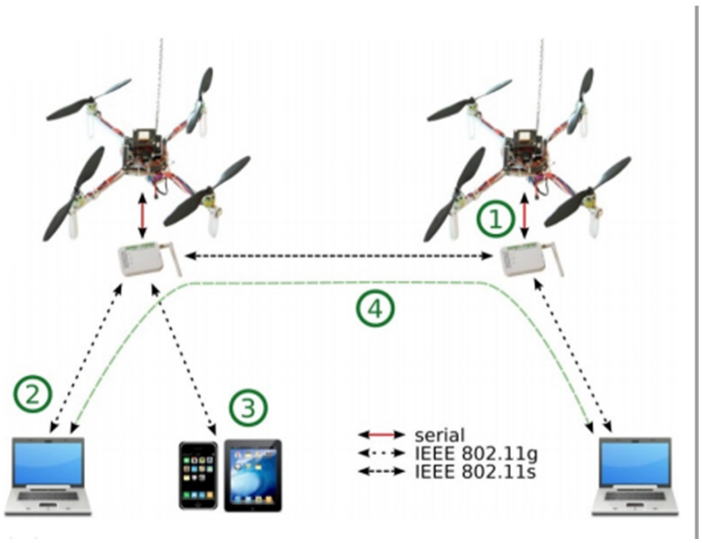

Figura 16: interconexión entre dos sistemas finales mediante dos nodos aéreos en el seno del proyecto UAVNet.

3 HARDWARE EMPLEADO

La frase más excitante que se puede oír en ciencia,
la que anuncia nuevos descubrimientos,
no es "¡Eureka!"sino "Es extraño…".

- Isaac Asimov –

En este capítulo se presenta, en primer lugar, una descripción genérica de los distintos equipos que conforman un multirrotor, así como una descripción específica de los equipos que se han embarcado en los multirrotores utilizados en este proyecto. Además, se explica cómo se ha realizado el montaje de los mismos y su puesta a punto tanto mecánica como en cuanto a configuración para realizar vuelos radiocontrolados.

3.1 Descripción del hardware embarcado

La primera fase de este proyecto consiste en ensamblar y ajustar un conjunto de multirrotores para poder efectuar vuelos de un modo "convencional", es decir, guiados por un piloto mediante control remoto. El objetivo es poder contar con una plataforma fiable y probada de vuelo sobre la que más adelante poder realizar pruebas y desarrollar tecnologías de intercomunicación entre UAVs. Cada uno de estos multirrotores constará, en esta primera fase, de los siguientes elementos:

- Chasis
- Motores
- Reguladores
- Hélices
- Placa de control y accesorios de ésta
- Receptor de radio
- Batería
- Power Distribution Board
- Alarma de voltaje de batería

A continuación se describen los elementos mencionados anteriormente.

3.1.1 Chasis

Es el soporte físico de todos los sistemas que van embarcados en el multirrotor. Consta de las bandejas superior e inferior, los brazos y el tren de aterrizaje. Los sistemas electrónicos y la batería se instalan sobre las bandejas. Los motores se instalan en los extremos de los brazos, los cuales deben ser lo suficientemente largos para evitar que las hélices colisionen entre sí. El chasis elegido mide 700mm entre extremos de brazos diagonales. Los reguladores suelen instalarse en los brazos también, para mantenerlos alejados de los sistemas electrónicos y evitar problemas de ruido eléctrico. La Figura 17: chasis sin ensamblar muestra los distintos elementos del chasis antes de ser ensamblados mientras que la Figura 18: chasis ensamblado muestra el chasis ya montado.

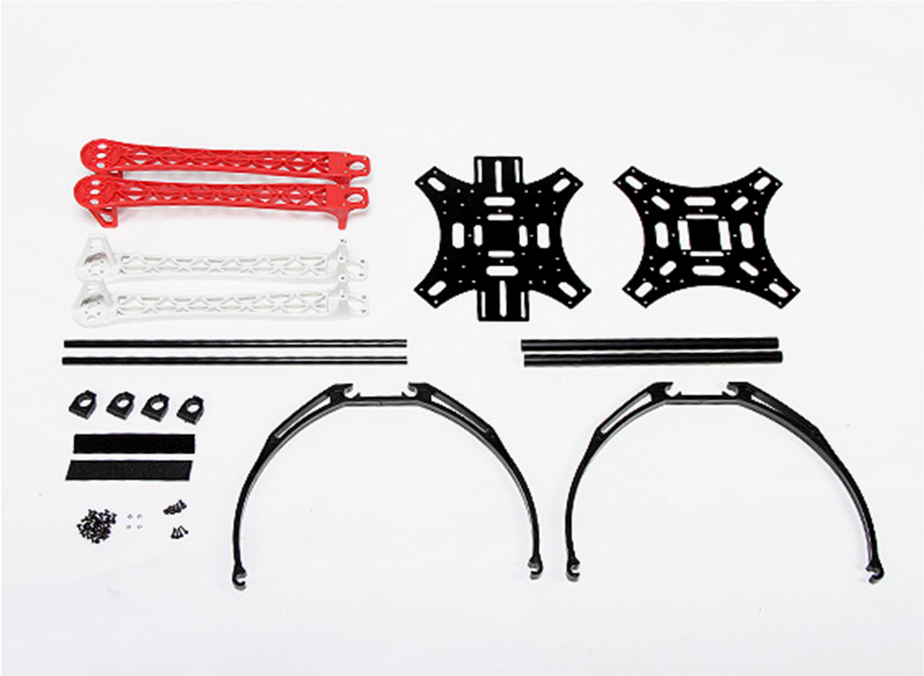

Figura 17: chasis sin ensamblar

En caso de instalar una cámara, ésta se instalará bajo la bandeja inferior, y el tren de aterrizaje la protegerá de impactar contra el suelo durante el aterrizaje. En caso de no instalar cámara no será necesario montar el tren de aterrizaje pues los brazos tienen unas "patas" en sus extremos para posarse en el suelo.

Figura 18: chasis ensamblado

3.1.2 Motores

Son los componentes que transforman la energía eléctrica proveniente de la batería en energía mecánica, para producir el giro de las hélices. En multirrotores se emplean motores trifásicos de tipo *brushless*, es decir, sin escobillas. Para usar estos motores se necesita de otros componentes, los reguladores, que serán explicados más adelante.

Figura 19: Motor MultiStar 4822

Para este proyecto se han elegido los motores MultiStar 4822 (Figura 19: Motor MultiStar 4822) [9] que dan una potencia máxima de 300 vatios. Sus características se muestran en la Tabla 4: Características de los motores. Cada motor incluye un conjunto porta-hélices, que se muestra en la Figura 20: Accesorios de los motores (conjunto porta-hélices).

KV (rpm/V)	490
Nº elementos Lipo	4 a 6
Corriente nominal	12 A
Corriente máxima	17 A (10 seg máx)
Resistencia interna	0.1 Ohm
Nº polos	22
Dimensiones	48 x 25 mm
Diámetro del eje	6 mm
Peso	98 g

Tabla 4: Características de los motores

Figura 20: Accesorios de los motores (conjunto porta-hélices)

3.1.3 Reguladores

Los reguladores, también comúnmente llamados ESCs (*Electronic Speed Controller*) son unos elementos electrónicos que tienen tres funciones distintas:

- Regulan la tensión de la batería de 14 voltios DC a 5 voltios DC para dar una alimentación adecuada a los sistemas electrónicos embarcados (función conocida como BEC o *Battery Eliminator Circuit*, ya que elimina la necesidad de utilizar una batería de 5V para alimentar la electrónica y otra de 14 para los motores).
- Convierten la corriente continua de la batería a corriente alterna trifásica, proporcionando a los motores la corriente que necesitan para girar.
- En función de la señal PWM (*Pulse Witdh Modulation* o Modulación por Ancho de Pulso) que reciben del receptor de radio, ajustan la velocidad de giro de los motores.

Figura 21: Regulador Turnigy PLUSH 30A

Para este proyecto se han seleccionado los reguladores Turnigy Plush 30A (Figura 21: Regulador Turnigy PLUSH 30A) que pueden aportar cada uno una corriente de 30A, claramente superior a la que exige cada motor (12A de corriente nominal cada motor). De este modo nos aseguramos de que no sufrirán ningún tipo de sobrecarga durante el vuelo. La Tabla 5: Características de los reguladores contiene las características de este modelo concreto de regulador.

Corriente máxima	30A
Corriente límite (pico)	40A
BEC	5V, 2A
Nº Elementos Lipo	2 a 4
Nº Elementos NiMH	5 a 12
Peso	25 g
Dimensiones	45 x 24 x 11 mm

Tabla 5: Características de los reguladores

3.1.4 Hélices

Las hélices son los elementos físicos que, al girar accionadas por los motores, provocan una corriente de aire en la dirección de su eje de giro. Cada multirrotor lleva 4 hélices, dos para giro en sentido de las agujas del reloj y otras dos para giro en sentido contrario a las agujas del reloj.

Figura 22: Hélices 14 x 4.7 pulgadas

Se han seleccionado unas hélices de fibra de carbono (Figura 22: Hélices 14 x 4.7 pulgadas) de 14 pulgadas (una pulgada equivale a 25.4 mm) de largo y 4.7 pulgadas de paso, para ejes de 5 a 6 mm de diámetro y con un peso de 20 gramos cada una de ellas. Que una hélice sea de 4.7 pulgadas de paso significa que la hélice avanzaría 4.7 pulgadas hacia delante al girar 360º alrededor de su eje si el aire fuese sólido y no hubiese ningún deslizamiento.

3.1.5 Placa de control y accesorios

La placa de control actúa como "cerebro" del sistema. Su principal misión es estabilizar el quadcopter y convertir las órdenes que recibe por radio en movimientos controlados y coherentes. Para este proyecto se ha optado por una placa compatible con el proyecto Ardupilot (Figura 23: Placa, módulos de telemetría, GPS y sensor de corriente), que es una familia de autopilotos de código abierto basados en la plataforma Arduino [10]. Ardupilot ha sido desarrollado por la empresa 3D Robotics y por una comunidad de desarrolladores de software libre. Esta tecnología permite convertir cualquier vehículo de alas fijas, alas rotatorias, o multirrotor (también barcos y coches) en un vehículo autónomo con capacidad de ejecutar misiones pre-programadas definidas por una serie de *waypoints* (puntos geográficos definidios por sus coordenadas), usando GPS. La placa elegida, la HKPilot Mega 2.7, incluye:

- HKPilot Mega 2.7 (Figura 24: Placa HKPilot Mega 2.7, sin carcasa).
- Módulo GPS Ublox LEA-6H con brújula integrada.
- Radios de telemetría bidireccionales en 915Mhz.
- Módulo medidor de consumo y alimentador (Figura 25: Módulo medidor de consumo y alimentador).
- Minim OSD para hacer vuelo en primera persona. Se trata de un pequeño dispositivo electrónico que combina la imagen proveniente de una cámara a bordo con la telemetría que proporciona la placa autopiloto y envía esta imagen combinada a un transmisor de vídeo. De este modo el UAV

puede ser pilotado en primera persona (FPV) visualizando el piloto la imagen en tiempo real proveniente del UAV así como información para facilitar la navegación, como puede ser la altura, rumbo, velocidad y dirección a seguir para llegar al punto de despegue. Este elemento no se ha utilizado en este proyecto, pues realizar vuelos en primera persona no forma parte de los objetivos.

Figura 23: Placa, módulos de telemetría, GPS y sensor de corriente

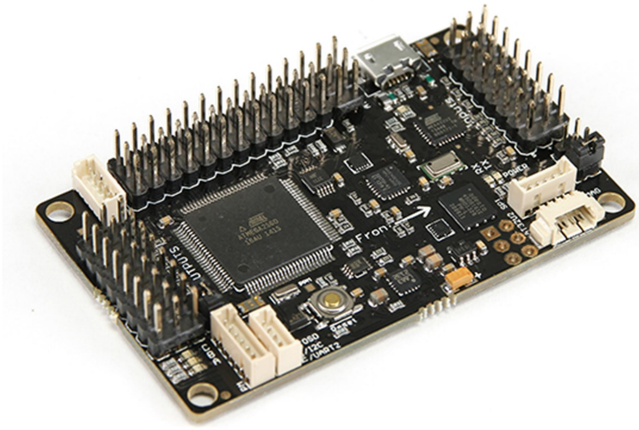

Figura 24: Placa HKPilot Mega 2.7, sin carcasa

Figura 25: Módulo medidor de consumo y alimentador

Algunas características de la placa de control son:

- Compatible con Arduino y ArduPilot Mega (APM)
- Brújula Honeywell HMC5883L-TR
- Acelerómetros/Giróscopos 6DOF MPU-6000
- Sensor barométrico MS5611-01BA03
- ATMEGA 2560 Y ATMEGA 32U-2 (MCU e integrado USB)
- Soporte para brújula externa
- Módulo GPS LEA-6h
- Puerto I2C
- Alimentador y medidor de consumo de hasta 10s

3.1.6 Receptor de radio

El receptor de radio es un elemento electrónico que recibe las órdenes que el piloto comanda al emisor de radio. En un aeromodelo convencional, el receptor se conecta directamente a los accionadores, es decir, a los servomotores que accionan las superficies de control y a los reguladores que controlan los motores. Sin embargo, debido a la complejidad de control de un quadcopter, el receptor se conecta a la placa de control, que en base a sus sensores, su programación y a las órdenes de radio que recibe decide cómo actuar sobre los reguladores. Cada multirrotor lleva un receptor Turnigy 9X8C-V2 de 8 canales (Figura 26: Receptor Turnigy 9X8C-V2).

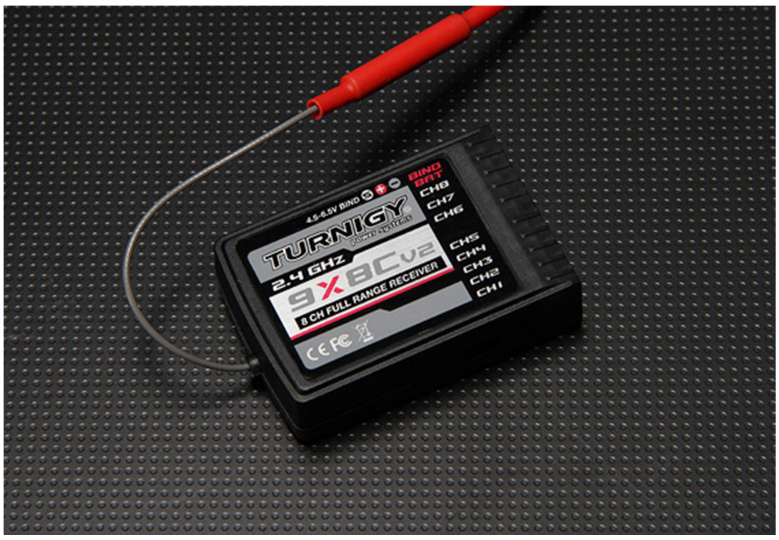

Figura 26: Receptor Turnigy 9X8C-V2

3.1.7 Alarma de voltaje de batería

La alarma de voltaje de batería es un sistema de seguridad, y debe ser el último recurso para conocer el estado de batería del multirrotor. Es un elemento que se conecta a la batería y controla el voltaje de cada elemento. Si alguno de ellos marca un voltaje inferior al voltaje mínimo recomendado por elemento, o bien si el conjunto de la batería marca un voltaje inferior al voltaje mínimo recomendado global, la alarma sonora se activa y avisa al piloto de que debe aterrizar inmediatamente. Dichos voltajes de alarma pueden ser programados. Cada uno de los multirrotores lleva un *HobbyKing 2-8s Cell Checker*, como el que puede verse en la Figura 27: HobbyKing Cell checker.

Figura 27: HobbyKing Cell checker

3.1.8 Power Distribution Board o PDB

Se trata de una pequeña placa PCB (*Printed Circuit Board* o Placa de Circuito Impreso), o en este caso un cable divisor, cuyo propósito es distribuir la corriente de la batería hacia los cuatro reguladores.

Figura 28: *Power Distribution Board*

Como puede apreciarse en la Figura 28: *Power Distribution Board*, el PDB tiene en uno de los extremos cuatro cables negros (negativo) y otros cuatro rojos (positivo) mientras que en el otro tiene un conector similar al de la batería. Además este modelo dispone de una salida extra para alimentar otro equipo, por ejemplo, un transmisor de vídeo.

3.1.9 Batería

La batería es el elemento que proporcionará la energía a todo el sistema, tanto para accionar los motores como para alimentar los sistemas electrónicos. Como se explicó previamente, los reguladores de los motores son los encargados de ajustar la tensión de la batería para alimentar los sistemas, así como de transformar la corriente continua en alterna para hacer girar los motores trifásicos.

La unidad elegida para este proyecto es una batería LiPO Zippy Flightmax 8000mAh, 4S1P 30C (Figura 29: Batería batería LiPO Zippy Flightmax 8000mAh, 4S1P 30C). A continuación se describe brevemente el significado de los distintos parámetros de la batería:

- **8000mAh**: hace referencia a la capacidad de la batería, que en este caso es de 8Ah. Podemos convertir este dato, teniendo en cuenta el voltaje de la batería, a Wh, que es una medida más convencional de energía eléctrica:

$$8\ Ah * 14{,}4\ V = 115{,}2\ Wh$$

- **4S1P**: está relacionado con la configuración interna de la batería, y significa que la batería está compuesta por un conjunto de 4 elementos LiPO en serie. Conociendo el voltaje nominal de un elemento LiPO (3,6V), este dato nos proporciona el voltaje nominal de la batería.
- **30C**: este dato hace referencia a la máxima descarga que puede soportar la batería, siendo 1C el equivalente a la capacidad nominal de la batería. Es decir, si la batería, de capacidad 8Ah, fuese de tipo 1C, la descarga máxima soportada sería de 8A. Al ser una batería de 30C, la descargar teórica máxima sería de 240A.

La Tabla 6: características de la batería muestra las características de la batería que ofrece el fabricante:

Capacidad	8000mAh
Voltaje	4S1P (14.8V)
Descarga	30C
Peso	845g
Dimensiones	166 x 69 x 35mm
Conector de equilibrado	JST-XH
Conector de descarga	5.5mm Bullet-connector

Tabla 6: características de la batería

Figura 29: Batería batería LiPO Zippy Flightmax 8000mAh, 4S1P 30C

3.2 Montaje y puesta a punto

En este subcapítulo se describen los procedimientos que se han seguido para instalar y configurar los distintos elementos de los multirrotores, así como el orden que se ha seguido para conseguirlo.

3.2.1 Montaje de los motores

El primer paso en el montaje de un multirrotor es instalar los 4 motores en los brazos del chasis. Para ello se emplearon 4 tornillos y unas arandelas. Es importante colocar los motores con los cables saliendo del motor por el lado del chasis, lo cual simplifica la sujeción de los mismos. Aunque en la imagen la hélice esté instalada en el motor, este paso no se dio hasta mucho más adelante. La Figura 30: Sujeción de los motores al chasis muestra un motor completamente instalado.

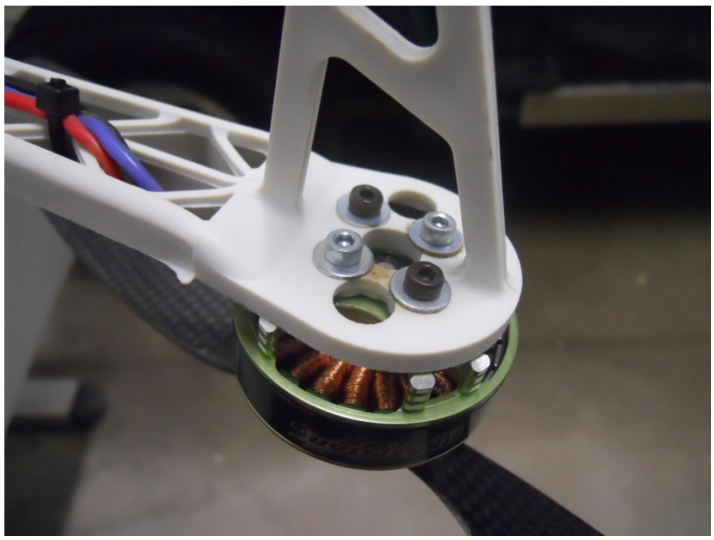

Figura 30: Sujeción de los motores al chasis

3.2.2 Emparejamiento del emisor y el receptor de radio

Para poder utilizar la radio, previamente el emisor y el receptor de radio deben estar emparejados, de forma que el receptor solo responda ante señales provenientes de la radio con la que ha sido emparejada. El procedimiento para emparejar la radio y el receptor es el siguiente:

- Encender la radio, pulsando el botón que se encuentra en su parte trasera, en el módulo transmisor.
- Conectar el cable de emparejamiento al receptor.
- Alimentar el receptor con una batería.
- Desconectar el receptor y el transmisor.

Este procedimiento solamente tiene que ser realizado una vez. Una emisora de radio puede estar emparejada con muchos receptores distintos, permitiendo utilizar una sola emisora para controlar muchos dispositivos. Sin embargo, un receptor únicamente puede estar emparejado con una emisora, por lo que habría que volver a realizar el procedimiento descrito anteriormente para poder utilizarlo con otra emisora distinta.

3.2.3 Instalación de los ESCs

Instalar los reguladores (Figura 31: Instalación de un regulador en un brazo del chasis) fue sin duda la operación individual que más tiempo ha requerido en el montaje de los multirrotores. La razón es que cada regulador tiene cinco cables de corriente, dos que provienen de la Power Distribution Board (de la batería) y tres que van a su motor correspondiente. Ninguno de ellos lleva conector, por tanto, a cada regulador hubo que soldarle 5 conectores (20 en total para cada dron).

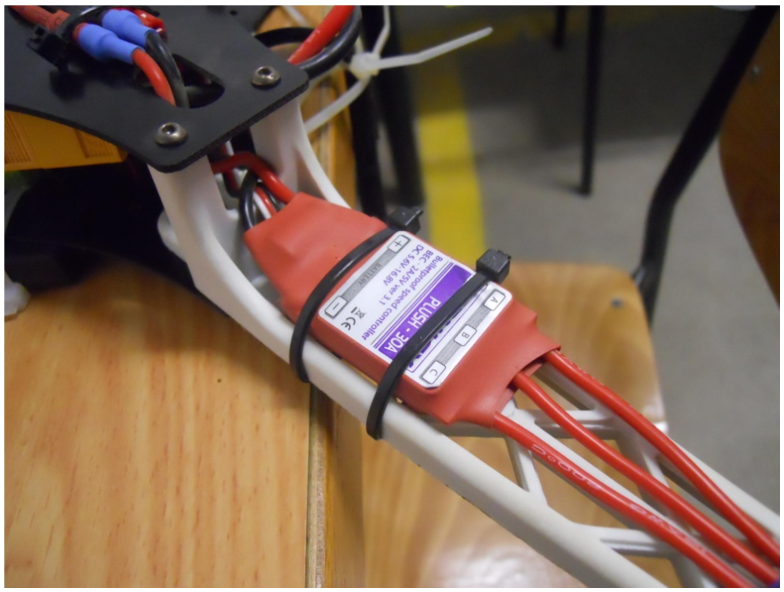

Figura 31: Instalación de un regulador en un brazo del chasis

Además, aunque los motores sí llevan conectores, cada motor lleva consigo una cantidad excesiva de cable, lo que implica desoldar dichos conectores, cortar el cable a la medida más adecuada y volver a soldarlos después. Esto implica seis soldaduras más en cada motor (24 contando los cuatro motores, 44 soldaduras en total para cada multirrotor).

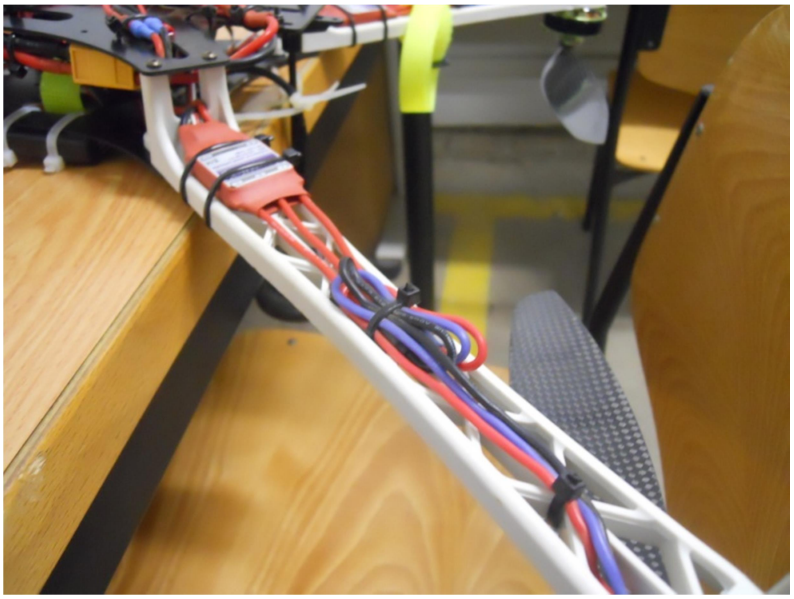

Figura 32: Instalación de un regulador y sus cables hacia el motor

Una vez el montaje eléctrico de cada regulador estuvo acabado, éstos fueron acoplados a la base de los brazos del chasis y sujetados mediante bridas de plástico. Los cables de cada regulador a su motor

correspondiente también fueron sujetados usando bridas de plástico (Figura 32: Instalación de un regulador y sus cables hacia el motor).

3.2.4 Instalación de la electrónica (controladora y receptor de radio)

En la Figura 33 podemos ver un diagrama que muestra cómo conectar la controladora al receptor, al GPS e incluso a un *gimbal,* una estructura autoestabilizada que suele utilizarse para sujetar una cámara.

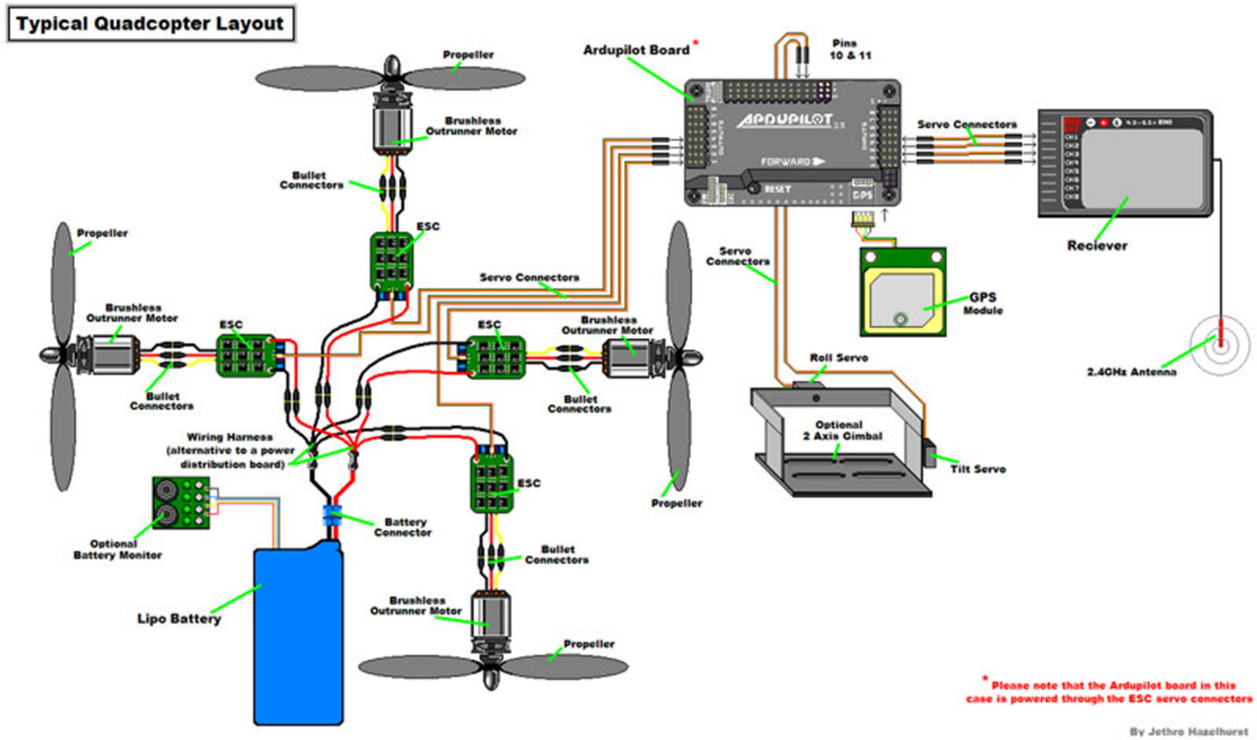

Figura 33: Conexionado de un multirrotor.

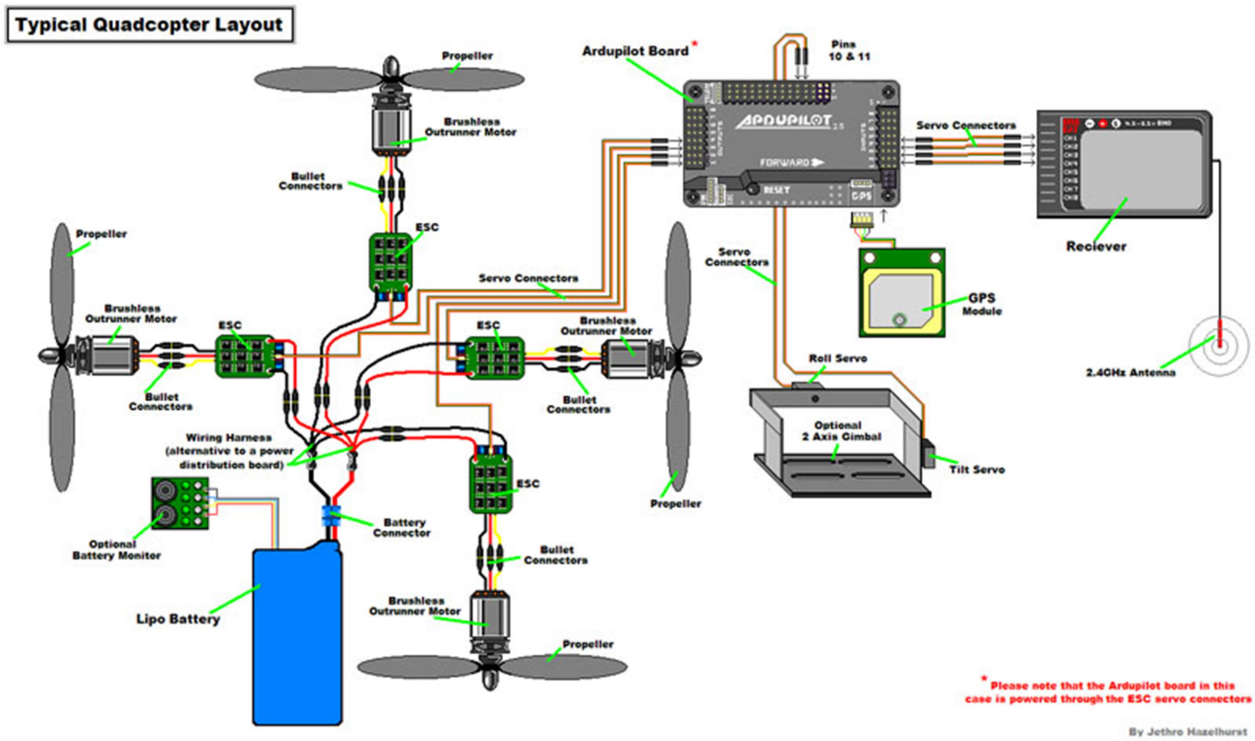

Figura 33: Conexionado de un multirrotor

En primer lugar, se procedió a instalar físicamente tanto el receptor de radio como la placa controladora. Una vez que el chasis está montado y atornillado, la bandeja inferior quedará prácticamente inaccesible. Debido a esto se tomó la decisión de colocar estos elementos en la bandeja inferior, dejando así la bandeja superior totalmente disponible para la batería.

Antes de poder instalar la controladora hubo que montarla, ya que ésta viene fuera de la carcasa. Además, nos da la opción de montarla sin carcasa por si pretendiésemos hacer un montaje extremadamente ligero. Una vez hecho esto, para fijar tanto el receptor como la placa al chasis se empleó una cinta adhesiva de doble cara, que viene incluida en el kit de la controladora.

Posteriormente se procedió a conectar el receptor con la placa controladora, utilizando cables típicos de servomotor (Figura 34: Cable de servo), que constan de tres hilos (negativo, positivo y señal).

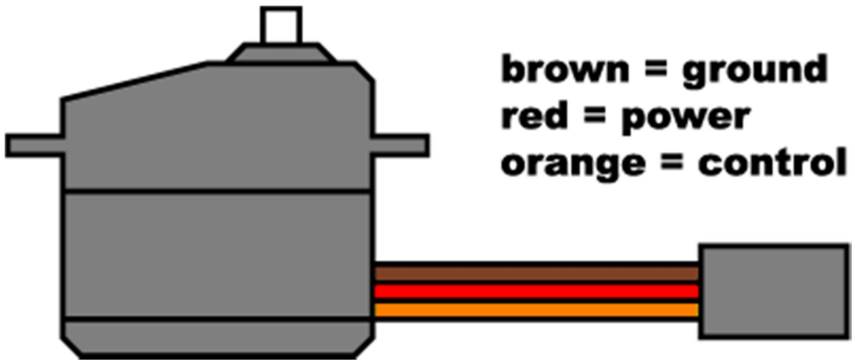

Figura 34: Cable de servo

En la Figura 35: Detalle del zócalo de entradas de la APM se puede ver indicado a qué canal corresponde cada una de las conexiones del zócalo de entradas de la placa. Esta asignación (canal 1 para Roll, 2 para Pitch, etc) se corresponde con la configuración tipo Modo 2 en los transmisores de radio. Esto quiere decir que si el transmisor está configurado en Modo 2 (es lo más habitual en la mayoría de países de Europa, no así en EEUU), podemos conectar cada canal del receptor con su correspondiente en la placa (canal 1 a entrada 1, canal 2 a entrada 2, etc).

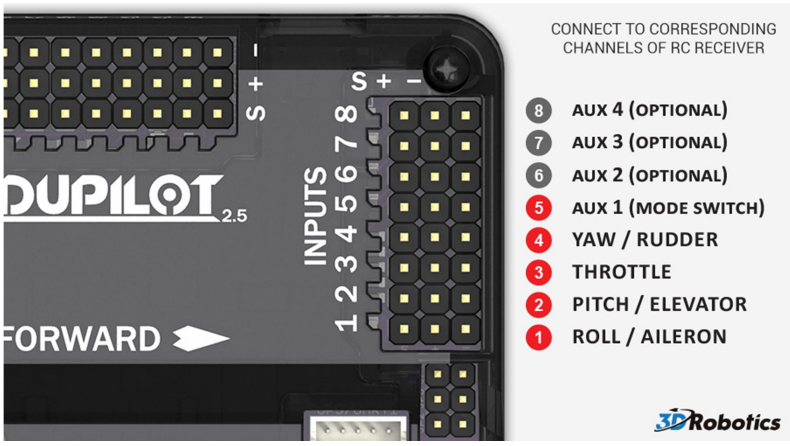

Figura 35: Detalle del zócalo de entradas de la APM

Como mínimo hay que conectar 5 cables (5 canales distintos de radio) entre la placa y el receptor, que permitirán controlar los 4 movimientos básicos de un multirrotor (Roll, Pitch, Throttle y Yaw), además de un canal extra para controlar el modo de vuelo en que se encuentra configurada la controladora. El resto de canales (la placa puede recibir hasta 8) pueden ser configurados para tareas auxiliares. En la Figura 36: Conexiones entre el receptor y la placa APM puede verse la conexión de los distintos canales entre el receptor de radio y la placa autopiloto.

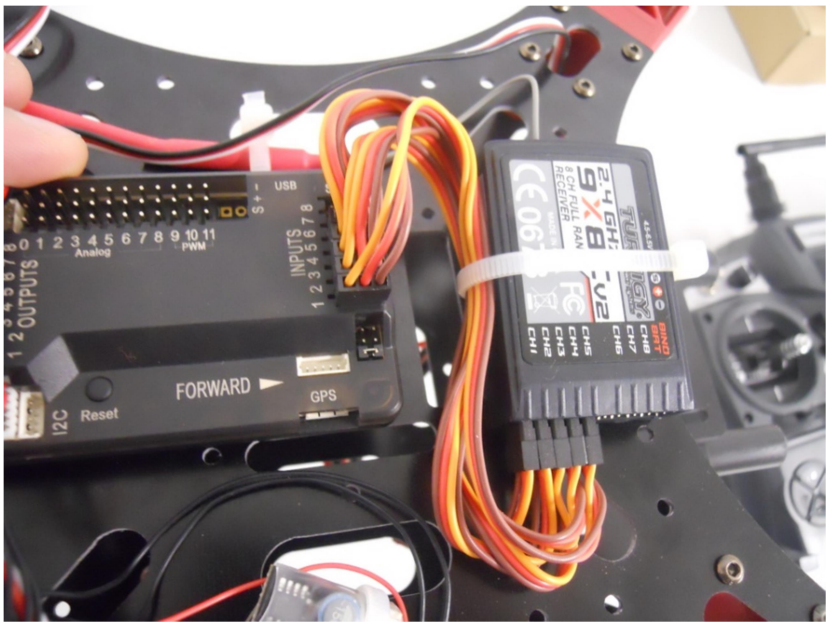

Figura 36: Conexiones entre el receptor y la placa APM

El siguiente punto es conectar los reguladores a la controladora. El tipo de cable para realizar esta conexión es el mismo que para conectar la controladora al receptor. En la Figura 37: Conexiones entre reguladores y APM en función de la configuración de vuelo se muestra un diagrama en el que puede verse el conexionado correcto.

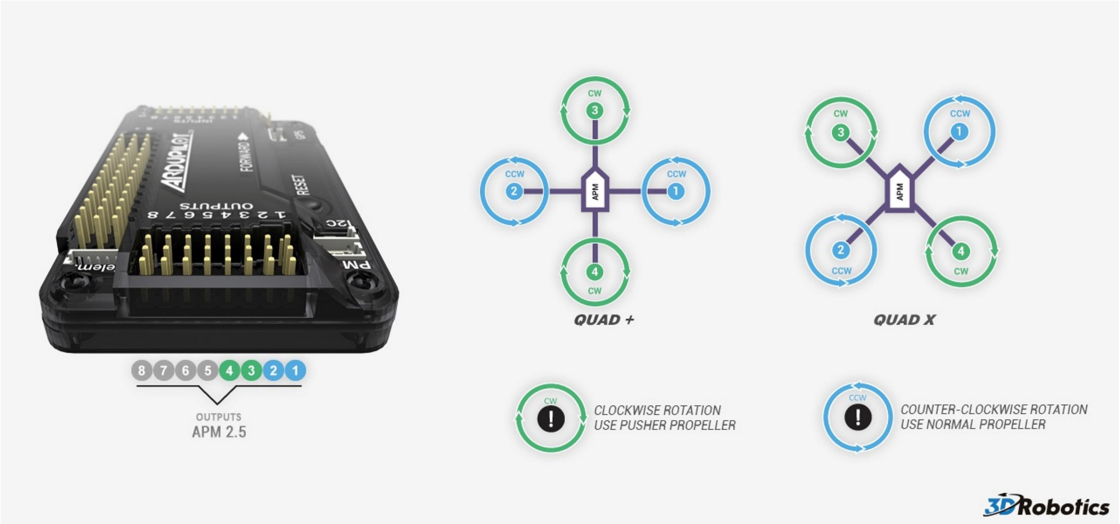

Figura 37: Conexiones entre reguladores y APM en función de la configuración de vuelo

La configuración elegida para este proyecto es la 'QUAD X' (al tener el chasis doble simetría podemos elegir tanto esa configuración como la configuración 'QUAD +'). En consecuencia, la conexión debe hacerse del siguiente modo:

- Output 1: Motor delantero derecho
- Output 2: Motor trasero izquierdo
- Output 3: Motor delantero izquierdo
- Output 4: Motor trasero derecho

Llegados a este punto hay que comprobar que los motores giran en la dirección adecuada, siguiendo el esquema mostrado en la Figura 38: Sentidos de giro de los motores, según configuración.

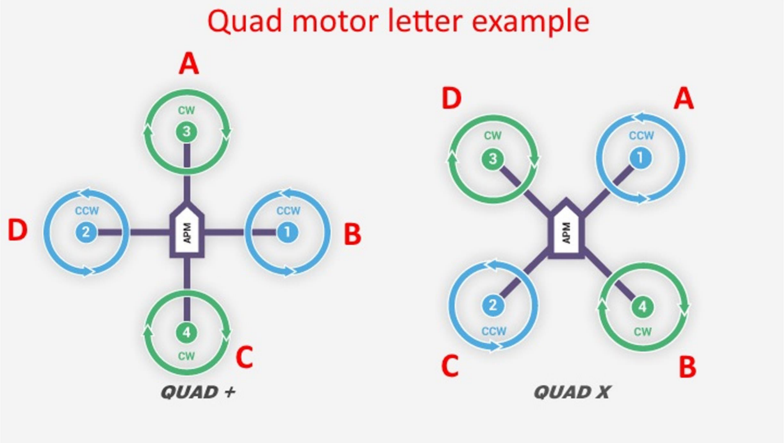

Figura 38: Sentidos de giro de los motores, según configuración

Como puede comprobarse, dos motores han de girar en sentido horario y dos en sentido anti-horario, de modo que el sistema esté equilibrado. Para comprobar el sentido de giro hay que realizar el siguiente procedimiento:

1. Asegurarse de no tener instaladas las hélices en los motores.
2. Conectar el transmisor de radio.
3. Conectar la batería del quadcopter.
4. "Armar" el quadcopter, manteniendo el stick izquierdo de la radio en posición atrás a la derecha durante cinco segundos.
5. Una vez que los motores empiezan a girar, comprobar el sentido de giro. Aquellos motores que no giren en el sentido adecuado pueden ser corregidos invirtiendo dos de los tres cables de alimentación del motor.

El siguiente paso será cargar la última actualización del firmware a la placa controladora. Para ello, lo primero será instalar en un PC la aplicación software Mission Planner. Una vez instalado, podemos conectar la controladora al PC mediante puerto USB (Figura 39: ubicación de la conexión USB en la APM).

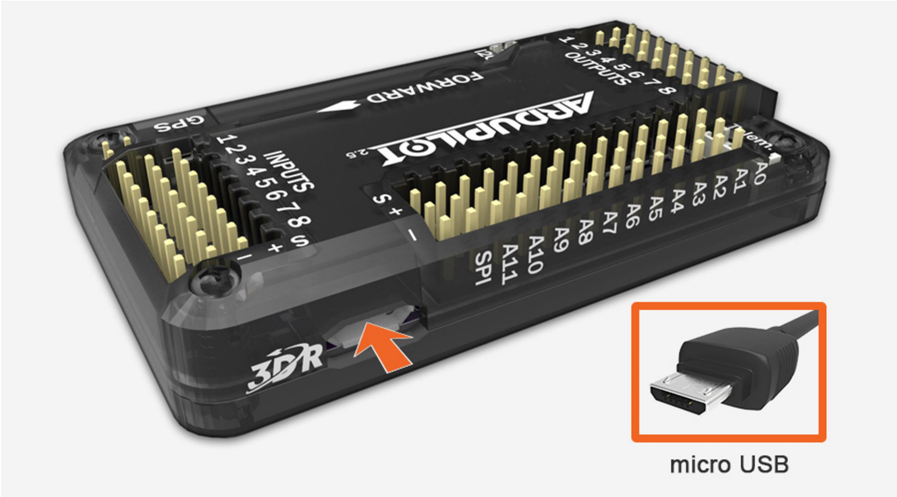

Figura 39: ubicación de la conexión USB en la APM

Windows debería detectar automáticamente la placa APM e instalar el driver de Arduino. Seguidamente, hay que abrir el programa Mission Planner y en el cuadrante superior derecho elegir "Arduino Mega 2560" y seleccionar una velocidad de transferencia de datos de 115200 baudios Figura 40: selección de puerto COM y "baudrate" en Mission Planner). No hay que presionar el botón "*Connect*" hasta que el firmware no esté instalado.

Figura 40: selección de puerto COM y "baudrate" en Mission Planner

Ahora hay que seleccionar el firmware que vamos a cargar en la APM, que dependerá del tipo de vehículo en el que vaya a ir montada la controladora. Dentro de la pestaña "*Install Firmware*", seleccionaremos el icono correspondiente a un quadcopter en configuración X (Figura 41: pantalla de selección de firmware en Mission Planner).

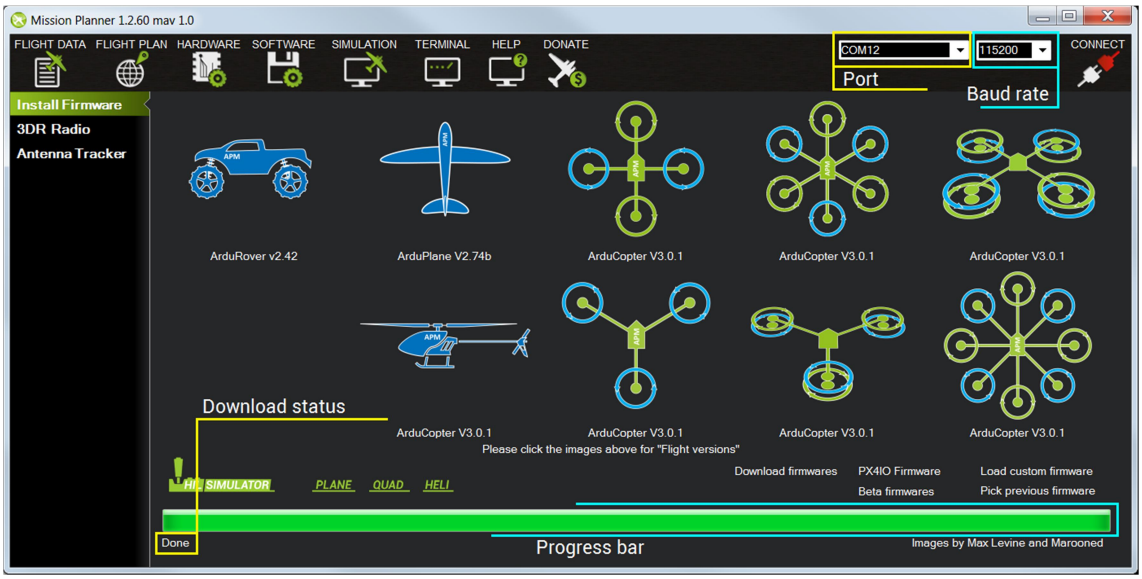

Figura 41: pantalla de selección de firmware en Mission Planner

Una vez que hemos seleccionado el tipo de vehículo, Mission Planner comprueba cuál es la última versión de firmware disponible y pide confirmación para instalarla. Hacemos click en "*Yes*" y esperamos a que el "*Download Status*" sea "*Done*". En ese momento, el firmware ha sido descargado e instalado en la placa.

Una vez el firmware ha sido instalado, podemos conectar el MissionPlanner con la controladora, pulsando el botón "*Connect"* en la esquina superior derecha (Figura 42: botones "Connect" y "Disconnect" de Mission Planner).

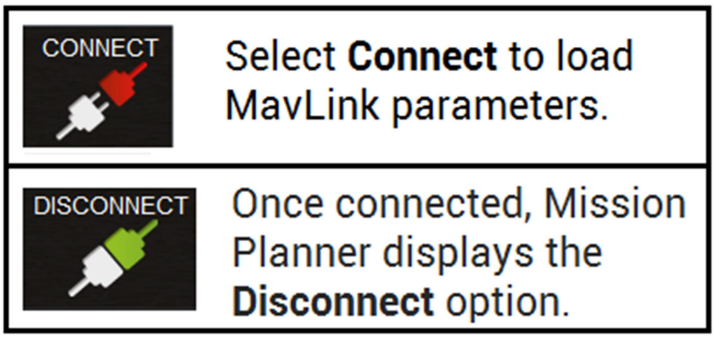

Figura 42: botones "Connect" y "Disconnect" de Mission Planner

Cuando en la esquina superior derecha aparece la opción "*Disconnect*", la conexión se ha realizado satisfactoriamente. La controladora APM tiene una gran cantidad de opciones, ajustes y configuraciones. Sin embargo, hay un grupo de opciones llamadas "*Mandatory Hardware Configuration*", que son una serie de configuraciones que tendremos que hacer de forma obligatoria para poder volar nuestro

multirrotor. Estos ajustes son los siguientes:

- Selección del tipo de chasis (el firmware instalado tiene soporte para varios tipos distintos, ahora hay que seleccionar el que emplearemos realmente)
- Calibración de la brújula
- Calibración de los acelerómetros
- Calibración de los controles de la radio

3.2.4.1 Configuración del autopiloto APM

Una vez toda la electrónica se encuentra convenientemente instalada en el drone, llega el momento de configurar la placa APM. Como se ha comentando anteriormente, Ardupilot, a diferencia de otras placas controladoras para drones, es un sistema que puede ser instalado en multitud de vehículos, ya sean terrestres (Figura 43: vehículo de tipología Rover, comandado por una APM) o aéreos. Por esta razón, es necesaria una perfecta configuración, de modo que el comportamiento del autopiloto se adapte a la tipología de vehículo seleccionada y a su comportamiento dinámico.

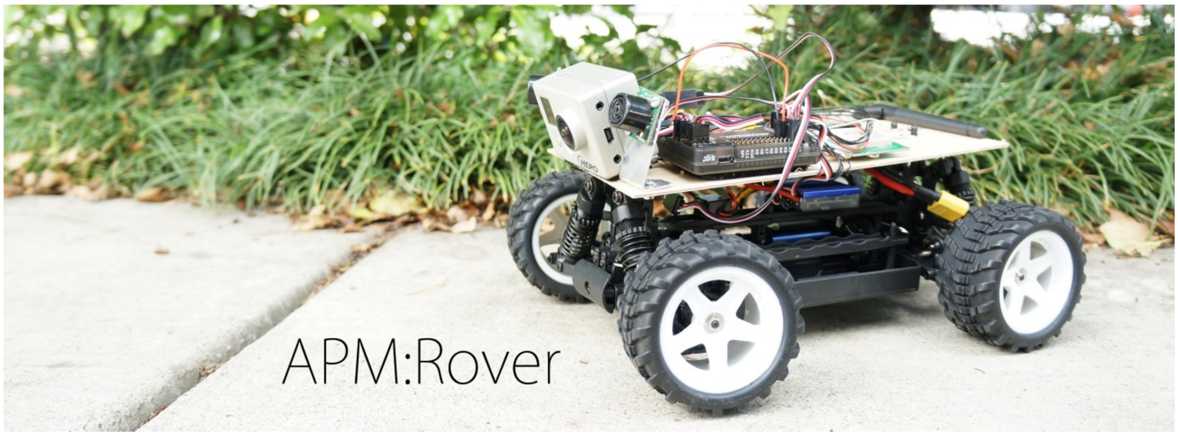

Figura 43: vehículo de tipología Rover, comandado por una APM

La configuración, al igual que la carga del firmware en la placa, se realiza con el programa MissionPlanner. APM distingue entre dos categorías de parámetros a configurar:

- *Mandatory Hardware Configuration*: son aquellos parámetros de obligada configuración antes de realizar el primer vuelo. Su correcta configuración es crucial para poder realizar un vuelo simple en modo manual.
- *Optional Hardware Configuration*: se trata de opciones que pueden mejorar el comportamiento del vehículo, así como dotarlo de nuevas posibilidades, pero que no son imprescindibles para realizar un vuelo manual.

Los ajustes considerados obligatorios, se describen a continuación, en los apartados siguientes.

3.2.4.1.1 Selección del tipo de chasis

Dentro de la pantalla *Initial Setup* de MissionPlanner, se hace click en *Mandatory Hardware* y posteriormente en *Frame Type*. De entre las opciones, elegimos el chasis en forma "X", que es el que corresponde con nuestra tipología de vehículo (Figura 44: selección de tipo de chasis en MissionPlanner). En el apartado Plus pueden elegirse otras configuraciones, incluyendo quadcopter en posición "+".

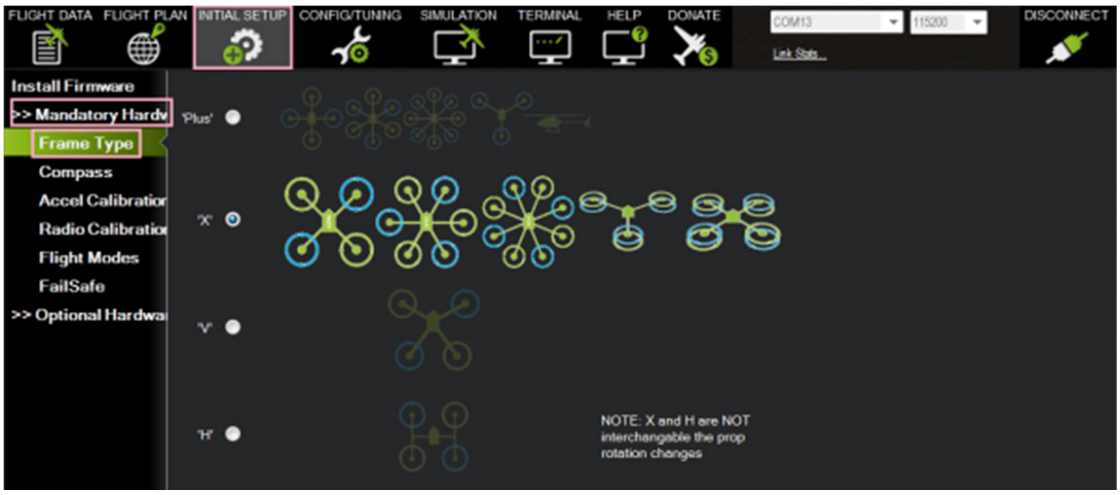

Figura 44: selección de tipo de chasis en MissionPlanner

3.2.4.1.2 Calibración de la brújula

El controlador APM contiene dos brújulas o magnetómetros: una incluida en la propia placa y otra en el módulo GPS. En un quadcopter lo habitual es utilizar la brújula del módulo GPS, pues ésta se encuentra elevada sobre el plano de los motores y esto implica una menor interferencia del campo magnético provocado por éstos sobre la brújula.

En la pantalla *Compass* (Figura 45: configuración de la brújula en Mission Planner), *e*n primer lugar, las casillas Enable y AutoDEC deben están seleccionadas. En el apartado *Orientation*, hay que elegir la opción APM with External Compass, que es la que se corresponde con nuestra configuración. Para que este ajuste funcione correctamente, no se puede cometer el error de no colocar el módulo GPS bien orientado. Éste tiene en su parte superior una flecha, que debe estar apuntando hacia la parte delantera del quadrotor.

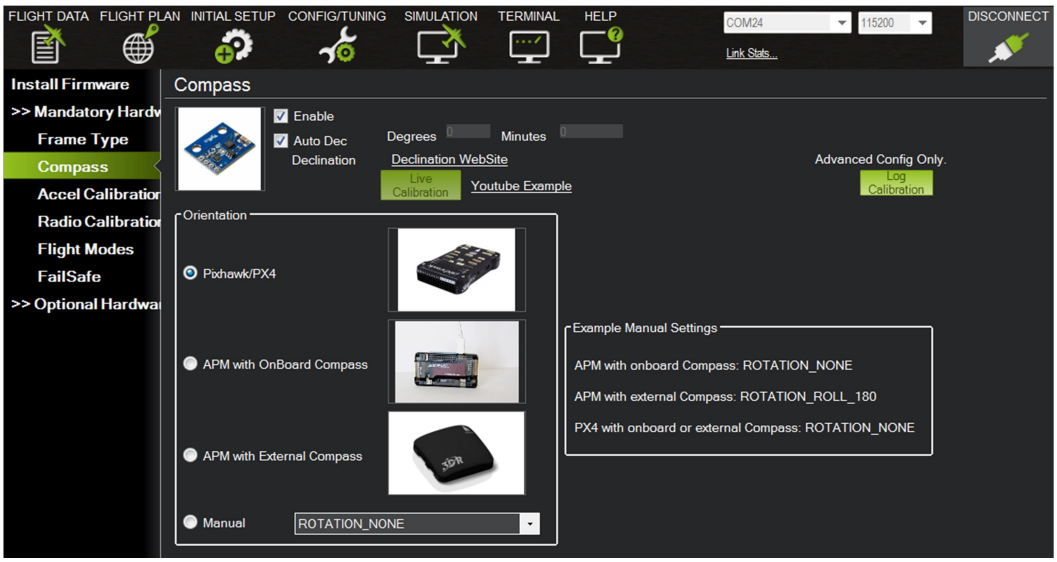

Figura 45: configuración de la brújula en Mission Planner

Ahora hay que pulsar el botón *Live calibration*. Entonces aparece una ventana informando de que disponemos de 60 segundos para hacer la calibración. Al pulsar *OK*, el proceso comienza. Durante los próximos 60 segundos hay que mover el quadrotor despacio en el aire de modo que cada una de sus caras (frontal, trasera, derecha, izquierda, superior e inferior) hayan apuntado hacia abajo en dirección al a teirra durante unos segundos (Figura 46: posiciones de calibración de brújula).

Figura 46: posiciones de calibración de brújula

Una vez el proceso ha terminado aparece una ventana informando de los *offset* de declinación magnética calculada. Se consideran buenos valores entre -150 y 150. Una vez presionamos *OK* de nuevo, el proceso de calibración de la brújula ha concluido.

3.2.4.1.3 Calibración de los acelerómetros

Para calibrar los acelerómetros, hay que seleccionar la opción *Accel Calibration*, en el menú *Initial Setup*. Este proceso requiere que el autopiloto tome lecturas en distintas posiciones del quadrotor, la cuales

pueden verse en la Figura 47: posiciones de calibración de los acelerómetros.

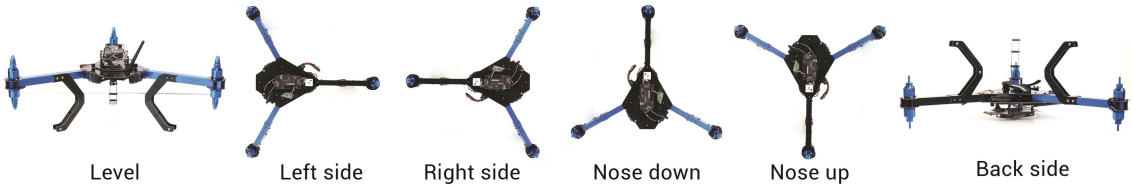

Figura 47: posiciones de calibración de los acelerómetros

La posición *level* es la que es más importante tomar con precisión ya que esa será la actitud de vuelo que el autopiloto considerará vuelo nivelado. Es importante mover el quadrotor inmediantamente después de pulsar la tecla en casa paso. Una vez se esté listo para calibrar, pulsar el botón *Calibrate Accel* (Figura 48: calibración de acelerómetros en MissionPlanner).

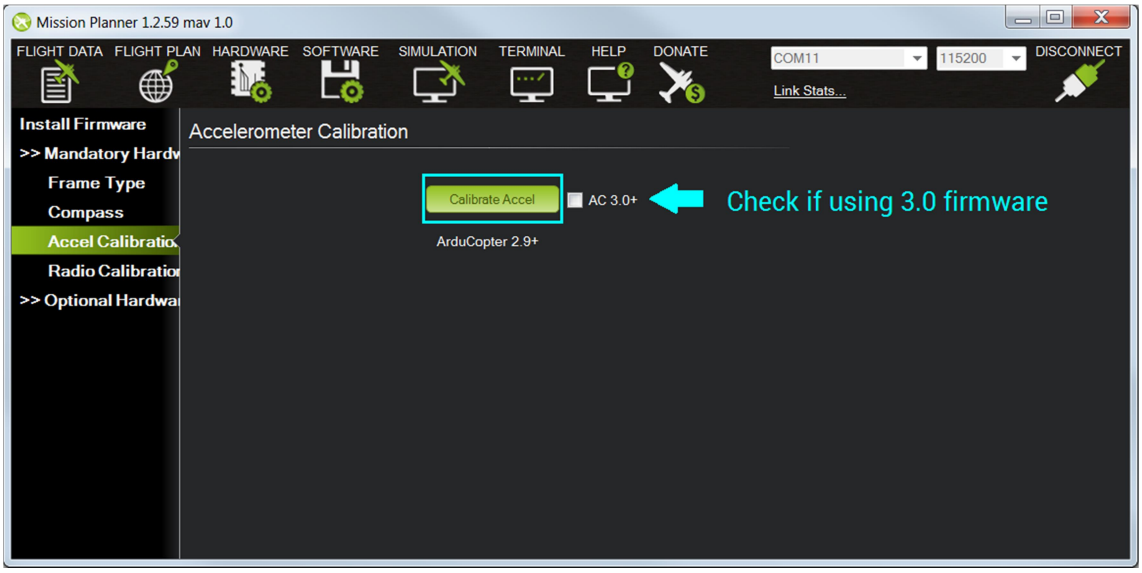

Figura 48: calibración de acelerómetros en MissionPlanner

El programa va indicando al usuario las posiciones en que debe ir colocando el quadrotor. Una vez el proceso de calibración ha terminado, aparece el mensaje *Calibration successfull*, como puede verse en la Figura 49: mensaje de calibración exitosa en MissionPlanner.

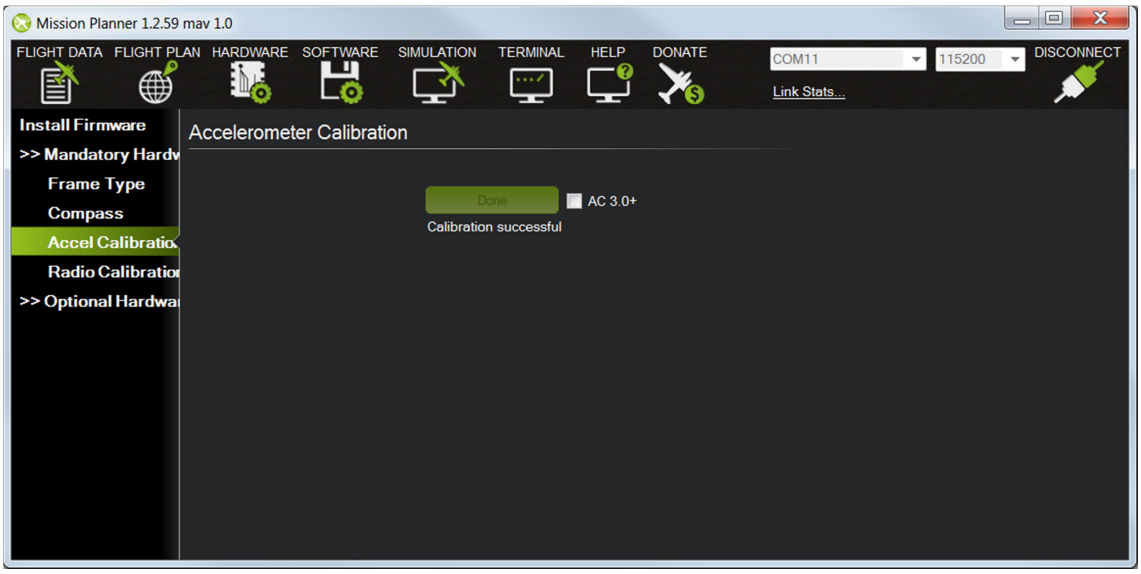

Figura 49: mensaje de calibración exitosa en MissionPlanner

3.2.4.1.4 Calibración de radio

El objeto de este punto es conseguir que el autopiloto reconozca adecuadamente la posición de los sticks de la radio con la que va a ser pilotado el quadrotor. No todas las radios modulan las posiciones de los *sticks* de forma exactamente igual. Además, las radios están dotadas de un sistema de *trims* para poder realizar pequeñas variaciones en el valor central de los canales de radio, y la posición de estos también varía de unas radios a otras. Una vez se complete este proceso, el autopiloto tendrá la información necesaria para conocer el valor de los distintos canales de radio, tanto cuando los sticks se encuentran en posición central como cuando se encuentran en los extremos. Los valores intermedios son obtenidos mediante interpolación de los valores extremos.

En primer lugar, hay que encender el transmisor de radio y verificar que el transmisor se encuentra en modo avión (independientemente del tipo de vehículo en el que esté montado la placa autopiloto, la radio debe estar funcionando en modo avión) y que los *trims* se encuentran en posición centrada.

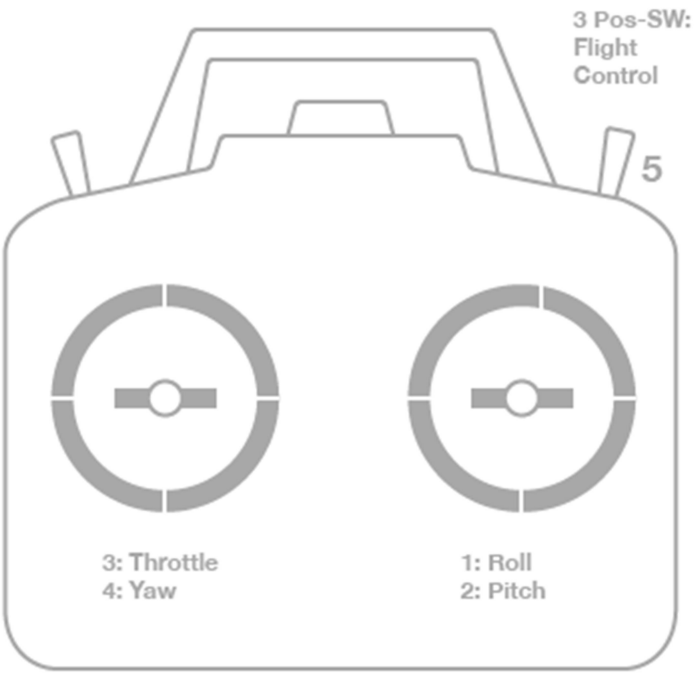

Figura 50: correspondencia de canales con los sticks en modo 2

Entre aeromodelistas y pilotos de UAVs existen dos configuraciones de *sticks* muy extendidas pero que difieren mucho la una de la otra. A un piloto habituado a operar en Modo 2 le resultaría muy complicado pilotar en Modo 1, y viceversa. Las dos configuraciones difieren en lo siguiente:

- En Modo 1, el *stick* izquierdo controla el ángulo de cabeceo (adelante y atrás) y el de guiñada (izquierda y derecha), mientras que el derecho controla el acelerador (adelante y atrás) y el alabeo (derecha e izquierda).
- En Modo 2 (Figura 50: correspondencia de canales con los sticks en modo 2), el *stick* izquierdo controla el acelerador (adelante y atrás) y la guiñada (izquierda y derecha), mientras que el derecho controla el cabeceo (adelante y atrás) y el alabeo (izquierda y derecha).

Las radios empleadas en este proyecto han sido configuradas en Modo 2, el más extendido en España. Independientemente del tipo de transmisor, un interruptor de tres posiciones de la radio debe ser configurado para controlar el canal 5, el canal de los modos de vuelo. En caso de disponer la radio de más de 5 canales, los restantes pueden ser utilizarse para funciones auxiliares.

Para proceder a la calibración, hay que ir a la opción *Calibrate Radio* (Figura 51: calibración de radio en MissionPlanner (1)), dentro del menú *Mandatory Hardware*. Mission Planner muestra ventana pidiendo al usuario que se asegure de que la radio está conectada, que el quadrotor tiene la batería desconectada (solo se alimenta el autopiloto a través de su cable USB) y las hélices no están puestas.

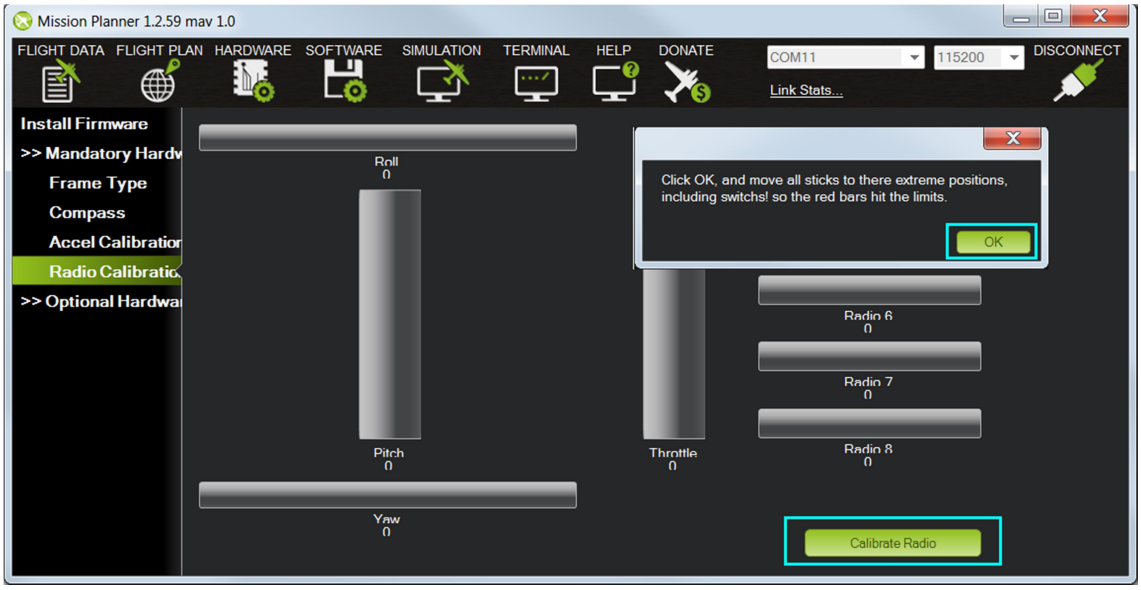

Figura 51: calibración de radio en MissionPlanner (1)

Una vez pulsamos OK el programa nos pide que movamos los sticks, así como los interruptores de los canales 5 y 6 a sus posiciones extremas, pudiendo observar estos movimientos en las barras de calibración (Figura 52: calibración de radio en MissionPlanner (2). Aparecen sobre éstas unas líneas rojas que indican los valores máximos y mínimos en cada canal. El movimiento de los sticks de la radio debería provocar los siguientes movimientos en las barras:

- Canal 1: mínimo = alabeo izquierda, máximo = alabeo derecha.
- Canal 2: mínimo = cabeceo hacia delante, máximo = cabeceo hacia atrás.
- Canal 3: mínimo = acelerador bajo, máximo = acelerador alto.
- Canal 4: mínimo = guiñada izquierda, máximo = guiñada derecha.

Figura 52: calibración de radio en MissionPlanner (2)

Cuando las barras rojas de guiñada, cabeceo, acelerador y alabeo así como del canal 5, estén fijas en sus valores máximos y mínimos, hacer click en *Click when Done*. Aparecerá un resumen de los datos de calibración (Figura 53: calibración de radio en MissionPlanner (3)). Valores alrededor de 1100 para mínimos y 1900 para máximos son habituales. Si alguna barra se mueve en dirección opuesta a la dirección en que se mueve el *stick*, esto indica que dicho canal está invertido en el transmisor de radio. En estos casos hay que recurrir a la función de inversión de canales de la radio para arreglarlo.

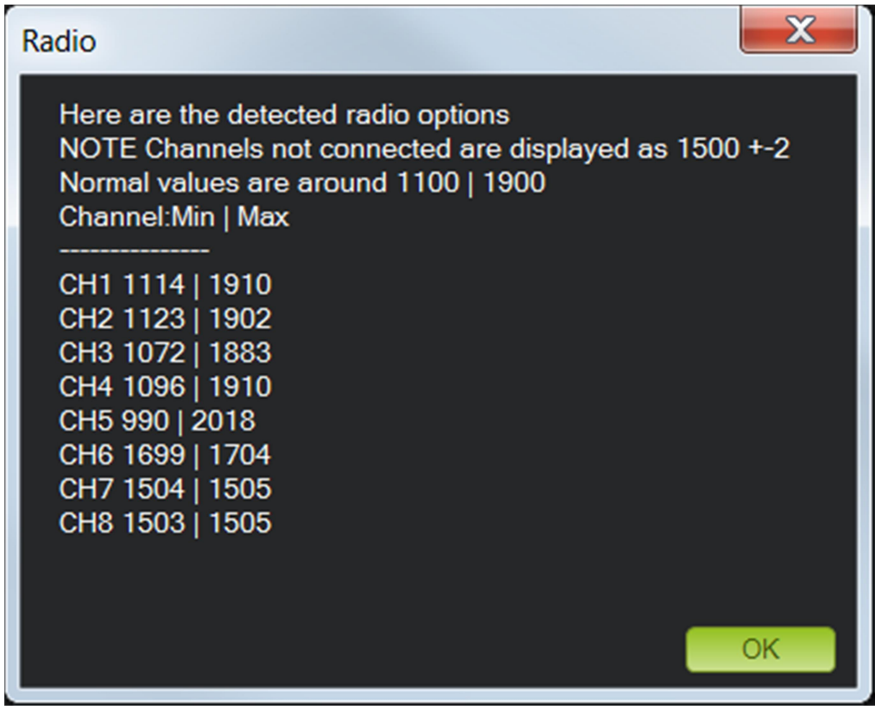

Figura 53: calibración de radio en MissionPlanner (3)

3.2.4.1.5 Calibración de los reguladores (ESCs)

Los reguladores son los encargados de hacer girar a los motores a la velocidad indicada por el autopiloto. Éstos deben calibrarse de tal modo que conozcan los valores mínimos y máximos de señal PWM que van a recibir del autopiloto.

Los reguladores están comandados por una señal de tipo PWM (*Pulse Width Modulation* o Modulación por ancho de pulso) la cual puede verse en la Figura 54: modulación PWM Como su propio nombre indica, el comando va codificado en el ancho de un pulso eléctrico, es decir, en la duración de dicho pulso. Normalmente esta señal se genera a 50hz. Entre pulso y pulso siempre hay un tiempo de 20ms, siendo el ancho del pulso de entre 1ms y 2ms, en función de las revoluciones de motor requeridas.

Figura 54: modulación PWM

Si bien este es el estándar, es posible que estos valores no sean exactamente los implementandos por nuestro equipo de radio, o bien mediante programación de la radio los valores de salida de los canales pueden estar alterados. Por este motivo la calibración y ajuste de los reguladores es un paso fundamental para que los motores y reguladores funcionen correctamente. Antes de realizar la calibración es necesario quitar las hélices. El procedimiento es el siguiente:

- Encender el transmisor de radio y poner el *stick* de acelerador al máximo (el procedimiento de calibración de la radio ha debido realizarse previamente).
- Conectar la batería del quadrotor. Los leds rojo, azul y amarillo se encenderán con un patrón cíclico. Esto quiere decir que el autpiloto está listo para entrar en modo de calibración de reguladores la próxima vez que se conecte.
- Con el *stick* de gases aún al máximo, desconectar y reconectar la batería. El autopiloto está ahora en modo de calibración de reguladores (los leds rojo y azul parpadearán alternativamente, como las luces de un coche de policía).
- Esperar a que los reguladores emitan primero un tono musical, luego un número determinado de pitidos indicando el voltaje de alimentación (3 si está alimentado con una batería de 3s, 4 si es de 4s, etc) y por último dos pitidos extra que indican que la calibración del punto de acelerador máximo ha concluido.

- Mover el *stick* de acelerador a la posición mínima. Los reguladores deberían emitir un tono largo, indicando que la calibración ha concluido. En este momento, los reguladores están armados y si se mueve ligeramente la palanca de acelerador, los motores comenzarán a girar.
- Por último, poner el acelerador al mínimo y desconectar la batería para abandonar el modo de calibración de reguladores.

Una vez realizdo el procedimiento anteriormente descrito, los reguladores han sido calibrados. No obstante, los reguladores tienen ciertos parámetros ajustables, cuya configuración, en función del dispositivo concreto, puede hacerse bien mediante el mando de acelerador o bien mediante algún tipo de dispositivo programador. Los Turnigy ESC 30A pueden configurarse de ambas formas, siendo la más cómoda usando una tarjeta programadora comercial. Los ajustes recomendados por la documentación de la plata autopiloto para los reguladores son los siguientes:

- Freno: desconectado
- Tipo de batería: Ni-xx (NiMH o NiCd). Aunque el multirrotor se alimente con una batería LiPO, se recomienda este ajuste para evitar que los reguladores paren los motores ante una bajada de voltaje puntual.
- Parada de motor: parada suave (por defecto).
- Límite de motor: bajo.
- Modo de arranque: normal (por defecto)
- Timing: medio.

3.2.4.1.6 Instalación del GPS

El módulo GPS 3DR uBlox con magnetómetro se conecta al autopiloto mediante dos conectores, uno correspondiente al GPS y otro al magnetómetro (Figura 55: conexionado entre el GPS + magnetómetro y la placa APM).

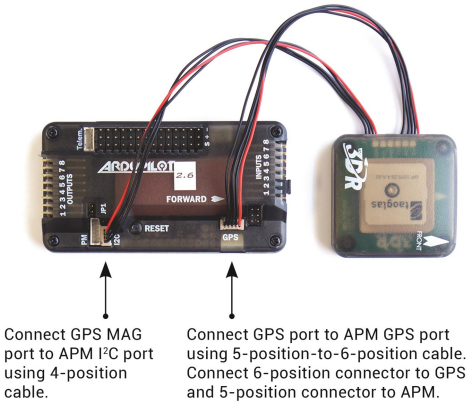

Figura 55: conexionado entre el GPS + magnetómetro y la placa APM

El modulo GPS, al estar separado del autopiloto, puede colocarse en un punto elevado del multirrotor, generalmente en un soporte a modo de "mástil", de forma que la cobertura GPS sea máxima y que el magnetómetro no se vea interferido por los campos magnéticos generados por los motores.

4 DESARROLLOS SOFTWARE

Si los extraterrestres nos visitaran, me daría vergüenza decirles que todavía extraemos combustibles fósles como fuente de energía.

- Neil deGrasse Tyson –

Los objetivos del presente proyecto, descritos en detalle en el capítulo 1, son el montaje de 3 multirrotores, el desarrollo de una aplicación para cargar una misión en el autopiloto APM y, por último, habilitar un sistema para intercambiar información entre multirrotores en vuelo. Este capítulo versa sobre todo lo relacionado con la consecución de los dos últimos objetivos: ampliación del hardware embarcado en cada UAV, descripción de las herramientas software utilizadas y exposición de los desarrollos de software realizados.

4.1 Ampliación del hardware embarcado

A lo largo del capítulo 3 se describen los distintos equipos que se han utilizado en el montaje de los multirrotores. Dicho equipamiento permite pilotar el multirrotor de forma manual y usar funcionalidades del autopiloto APM, siendo algunas de ellas las siguientes:

- Pilotaje manual
- Vuelo estacionario en un punto geográfico concreto (Modo *Loiter*)
- Seguimiento de una sucesión de puntos geográficos preconfigurados con la aplicación MissionPlanner (Modo *Auto*)

A partir de este momento, al hardware descrito en el capítulo 3 se le llamará *equipamiento básico*. A dicho equipamiento básico se le han añadido dos artículos más, con el objeto de ampliar las funcionalidades de los multirrotores. Dichos equipos son:

- Ordenador de placa única, Raspberry Pi (Figura 56: ordenador de placa única Raspberry Pi 2 Model B)
- Módulo de transmisión Wi-Fi.

A la combinación del equipamiento básico y los equipos anteriores se le llamará *equipamiento ampliado*.

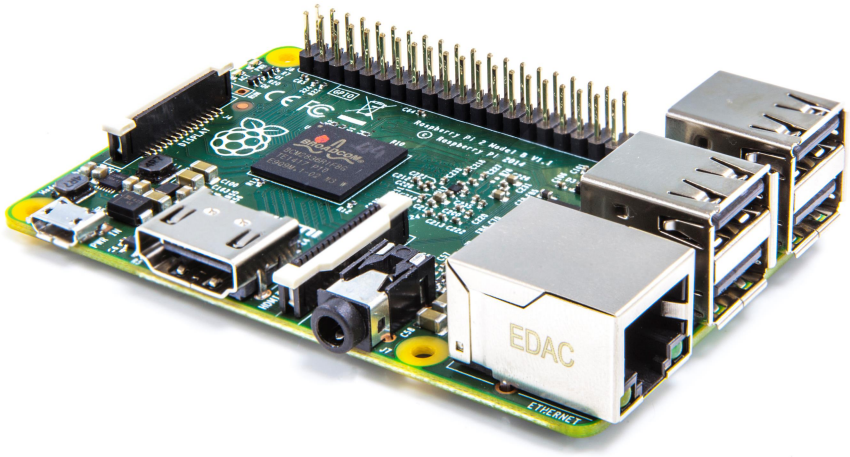

Figura 56: ordenador de placa única Raspberry Pi 2 Model B

El firmware del autopiloto no ha sido modificado. Las funcionalidades extra se han obtenido mediante la comunicación de la placa Raspberry Pi con el autopiloto (mediante cable USB) y con otros dispositivos (mediante la conexión Wi-Fi inalámbrica).

4.1.1 Raspberry Pi 2 Model B

La tarjeta Raspberry Pi [11] es un ordenador de placa reducida (SGB, Single Board Computer) de bajo precio y del tamaño de una tarjeta de crédito. Puede onectarse a un monitor, un teclado y un ratón y utilizarse como un ordenador de sobremesa convencional, si bien este no es su propósito principal. Debido a su reducido precio y su versatilidad, distintos modelos de Raspberry Pi son utilizados en proyectos de robótica amateur (Figura 57: robot terrestre controlado mediante Raspberry Pi).

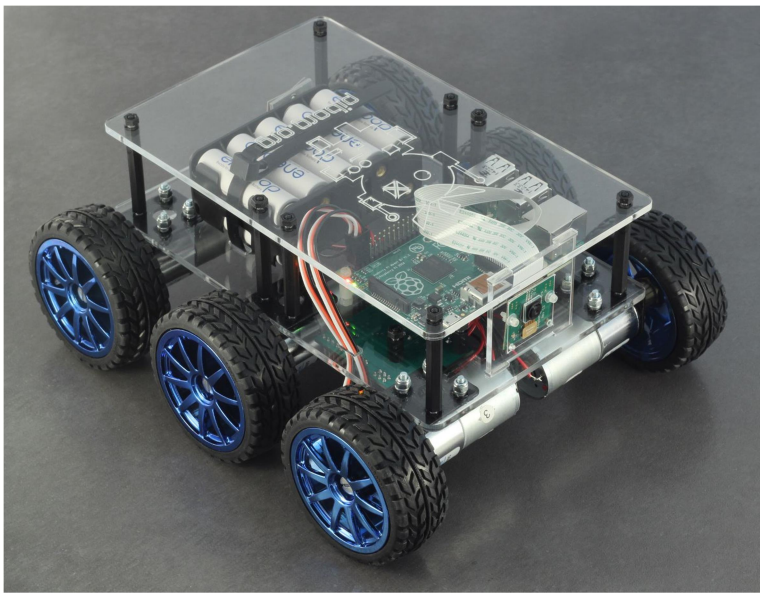

Figura 57: robot terrestre controlado mediante Raspberry Pi

Algunas otras aplicaciones típicas en que se utiliza Raspberry Pi son servidores de almacenamiento de datos o como HTPC (*Home Theater Personal Computer*, ordenador de cine en casa), dada su capacidad para reproducir vídeo en alta resolución.

El modelo Raspberry Pi 2 Model B (Figura 56: ordenador de placa única Raspberry Pi 2 Model B) tiene las siguientes características:

- Procesador ARM Cortex-A7, 4 núcleos, 900MHz
- 1GB memoria RAM
- 4 puertos USB
- 40 pines GPIO (*General Purpose Input & Output*, entrada/salida de propósito general)
- Puerto HDMI (*High-Definition Multimedia Interface* o interfaz multimedia de alta definición)
- Puerto Ethernet
- Jack audio y vídeo compuesto, 3.5mm
- Interfaz para cámara (CSI)
- Interfaz para display (DSI)
- Ranura tarjeta Micro SD
- Núcleo gráfico VideoCore IV 3D

Al disponer de un procesador con arquitectura ARMv7, pueden utilizarse distribuciones GNU/Linux para ARM.

4.1.2 Módulo Wi-Fi WiPi

El modulo Wi-Fi Wi-Pi (Figura 58: Módulo Wifi Wi-Pi) es un dispositivo Wi-Fi USB preparado para conectar una Raspberry Pi a una red Wifi local. Utiliza tecnología 802.11n y admite velocidades de 150Mbps.

Figura 58: Módulo Wifi Wi-Pi

Algunas de sus características son:

- Antena integrada
- IEEE 802.11n (compatible con IEEE 802.11g y IEEE 802.11b)
- Rango de frecuencias entre 2.4Ghz y 2.4835Ghz
- Canales 1 a 13
- Potencia máxima de transmisión de 20dBm
- Soporte nativo con la distribución Raspbian Wheezy
- USB 2.0
- Encriptación WEP 64bit/128bit/152bit
- WPA-PSK/WPA2-PSK
- WPA/WPA2

A continuación se muestra una imagen (Figura 59: equipamiento ampliado) con los elementos que componen el *equipamiento ampliado,* obviando la batería y la alarma de voltaje de batería.

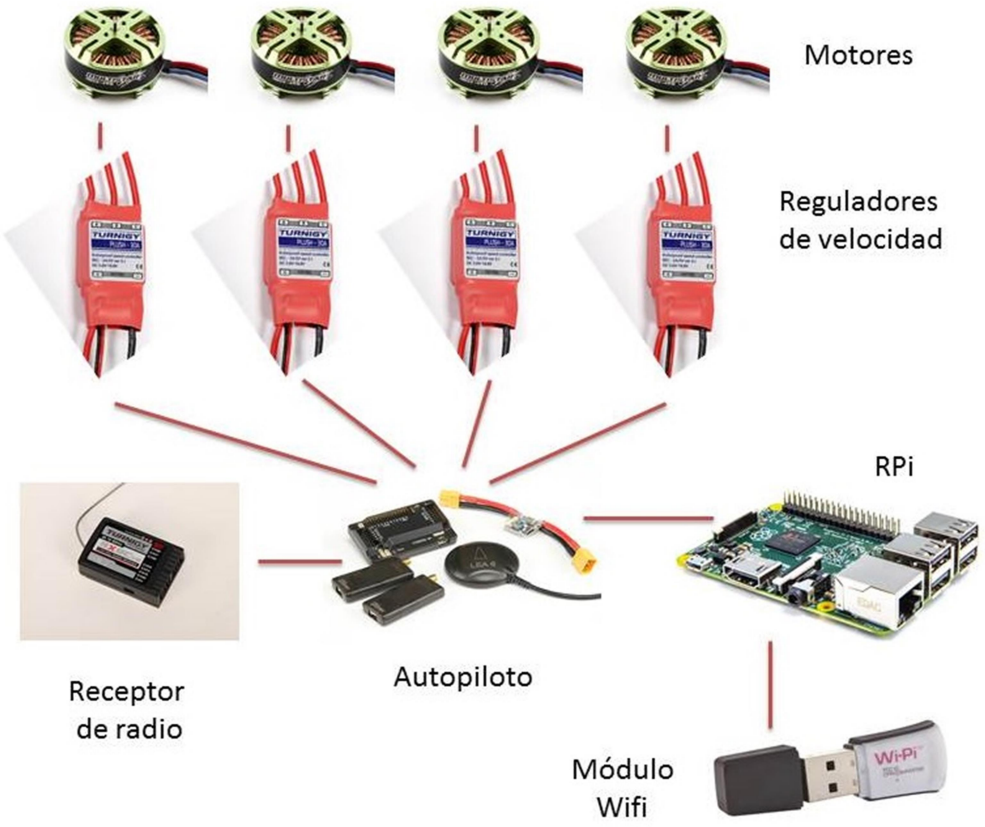

Figura 59: equipamiento ampliado

La conexión entre la Raspberry Pi y el autopiloto se realizó mediante un cable USB, del mismo modo que se conectaría el autopiloto a un ordenador de sobremesa.

4.2 Descripción de herramientas de software empleadas

En este subcapítulo se describen brevemente cuáles han sido las principales herramientas software empleadas en los desarrollos del proyecto.

4.2.1 Python

Los distintos desarrollos software de este proyecto se han llevado a cabo en lenguaje Python, utilizando distintos módulos cuando ha sido necesario. Algunas de las características principales del lenguaje Python son las siguientes [12]:

- Es un lenguaje multipropósito con el que pueden hacerse scripts, aplicaciones con interfaz gráfica o incluso desarrollos web.
- Es multiplataforma, habiendo versiones disponibles de Python en muchos sistemas informáticos distintos.

- Es multiparadigma, permitiendo combinar programación orientada a objetos con programación orientada a procedimientos.
- Es interpretado, es decir, el código no tiene que ser previamente compilado. Esta característica aumenta la velocidad de desarrollo pero, en contrapartida, reduce la velocidad de ejecución del software. En algunos casos, cuando se ejecuta por primera vez un código, se producen unos códigos que se guardan en el sistema y que aceleran la ejecución próxima del software.
- Es dinámicamente tipado, es decir, una misma variable puede tomar valores de distinto tipo en distintos momentos.
- Es interactivo. Dispone de un intérprete por línea de comandos en que pueden introducirse sentencias y ejecutarlas una a una, ayudando así al proceso de desarrollo.
- Dispone de funciones y librerías incorporadas para el tratamiento de strings, números, archivos, etc. Además existen muchas librerías que son fácilmente importables a proyectos nuevos.
- Posee una sintaxis muy clara, ya que las distintas porciones de código se separan mediante identaciones (márgenes) de obligado cumplimiento. Esta forma de separar subconjuntos de código resulta más intuitiva que la utilización de llaves propia de C.
- Es totalmente gratuito para uso personal, empresarial o académico.

4.2.2 MavLink

MavLink (Micro Air Vehicle Link) es un protocolo de comunicación muy utilizado por distintos autopilotos, incluido ArduPilot, para comunicarse con una estación de control en tierra (GCS).

Un mensaje MavLink es una cadena de bytes, codificada por el autopiloto o la GCS y enviada al otro dispositivo a través de un puerto serie (cableado o inalámbrico). Dicha codificación consiste en colocar el mensaje en una estructura de datos adecuada y enviarla, añadiendo mecanismos de detección de errores [13].

Un mensaje MAVLINK tiene un tamaño de 17 bytes, y tiene una estructura consistente en 6 bytes de cabecera, 9 de carga de pago y 2 bytes de seguridad:

- 6 bytes de cabecera
    - 0. Cabecera del mensaje. Siembre es 0xFE.
    - 1. Longitud del mensaje
    - 2. Número de secuencia
    - 3. System ID. Identificación del sistema que envía el mensaje.
    - 4. Component ID. Componente del sistema que envía el mensaje.

- 5. Identificación del mensaje.

- Payload (carga de pago, la información que se desea transmitir).
- 2 bytes de seguridad (Checksum), para detectar y desechar mensajes corruptos.

El software, ya sea el software GCS o el software autopiloto, comprueba que el mensaje recibido es válido (mediante los bytes de seguridad). Para evitar que se produzcan muchos mensajes defectuosos, la velocidad de transferencia de datos suele reducirse de 115200 baudios a 57600 baudios cuando el puerto serie es inalámbrico.

Durante el funcionamiento nominal de un sistema Drone-GCS, uno de los dos dispositivos recibe un mensaje a través de la interfaz serie y lo decodifica. De dicho mensaje se extraerá la carga de pago, así como la identificación del mensaje, para saber qué representa la información del *Payload*. Por último esta información, ya identificada, se coloca en su estructura de datos correspondiente. Hay muchas de ellas, por ejemplo para información de actitud, coordeandas GPS, etc. Estas estructuras son idénticas en todos los sistemas que implementan MavLink, haciéndolas compatibles entre sí.

4.2.3 pyMavlink

Se trata de un módulo para Python, para gestionar el protocol MavLink a bajo nivel.

4.2.4 DroneKit

Dronekit [14] es una API (*Application Programming Interface* o Interfaz de Programación de Aplicaciones) creada por 3DRobotics (empresa creadora de los autopilotos APM y PixHawk). Posee métodos para gestionar el protocolo MavLink a un nivel más alto que pyMavlink, y ofrece soporte para más lenguajes (Android e iOS). Además, la documentación es extensa y hay multitud de ejemplos disponibles.

4.2.5 Tkinter

Es una herramienta básica para realizar Interfaces Gráficas de Usuario (GUIs) en python. Este módulo está incluido por defecto en las versiones de Python para Linux y Windows [15].

4.3 Desarrollos software en Python

En este subcapítulo se describen los desarrollos software que se han realizado en este proyecto, persiguiendo cubrir los objetivos planteados en el capítulo 1. Dichos desarrollos han sido realizados en Python.

4.3.1 Mission Loader

4.3.1.1 Concepto

El concepto de Mission Loader consiste en una aplicación en Python cuyo propósito es cargar una misión en el autopiloto APM, de forma que el multirrotor ejecute dicha misión posteriormente y de un modo autónomo, sin intervención del piloto. Dicho concepto coincide exactamente con el primero de los objetivos de este proyecto.

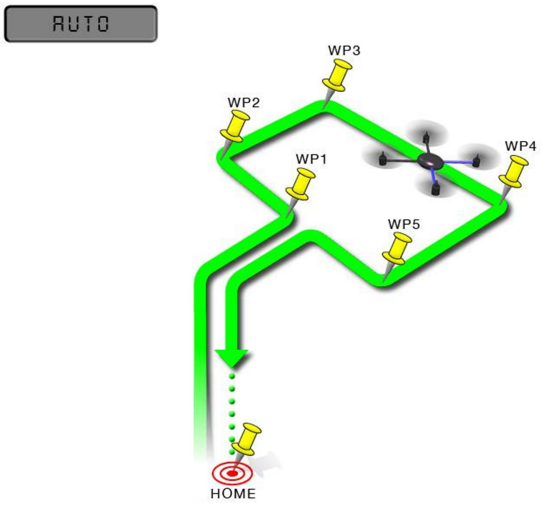

Figura 60: quadrotor siguiendo una misión definida mediante *waypoints*

El pilar principal de esta idea reside en uno de los modos de funcionamiento del autopiloto APM, llamado modo Auto (Figura 60: quadrotor siguiendo una misión definida mediante *waypoints*). Cuando el quadcopter se encuentra en modo Auto, este ejecuta una misión preprogramada, que previamente ha debido ser almacenada en su memoria interna. Dicha misión se compone de comandos de navegación (*waypoints*) y comandos de acción como por ejemplo *takeoff* (despegue), *land* (aterrizaje) y RTL (*Return to Launch Site* o vuelta a casa).

Para comenzar una misión, el piloto debe armar el quadcopter, activar el modo Auto y poner la palanca del acelerador en la posición central. En el momento en que el autopiloto detecta un incremento en la posición del acelerador, la misión comienza. En caso de que el piloto active el modo Auto con el quadcopter en vuelo, la misión comenzará omitiendo la orden de despegue, pues carece de sentido al estar el quadcopter ya en vuelo. En cualquier momento el piloto podría retomar el control manual, activando, por ejemplo, el modo Stabilize. Si se vuelve a activar el modo Auto, la misión se reinicia, desde el primer comando.

El último comando de una misión debería siempre ser o bien del tipo RTL (Return To Launch, el quadcopter vuelve al punto donde despegó y aterriza) o bien LAND (aterrizaje en el último waypoint), para asegurar que la misión termina exitosamente. De lo contrario, el piloto tendrá que tomar el control del quadcopter al terminar la misión, y aterrizarlo manualmente.

Una vez que el quadcopter entra en contacto con el suelo, el piloto debe colocar el acelerador en la posición mínima, el quadcopter se desarmará y la misión habrá concluido.

Las operaciones de diseño y carga de una misión pueden hacerse con distintos software GCS, siendo MissionPlanner (Figura 61: diseño de una misión con el software MissionPlanner) el más conocido de ellos.

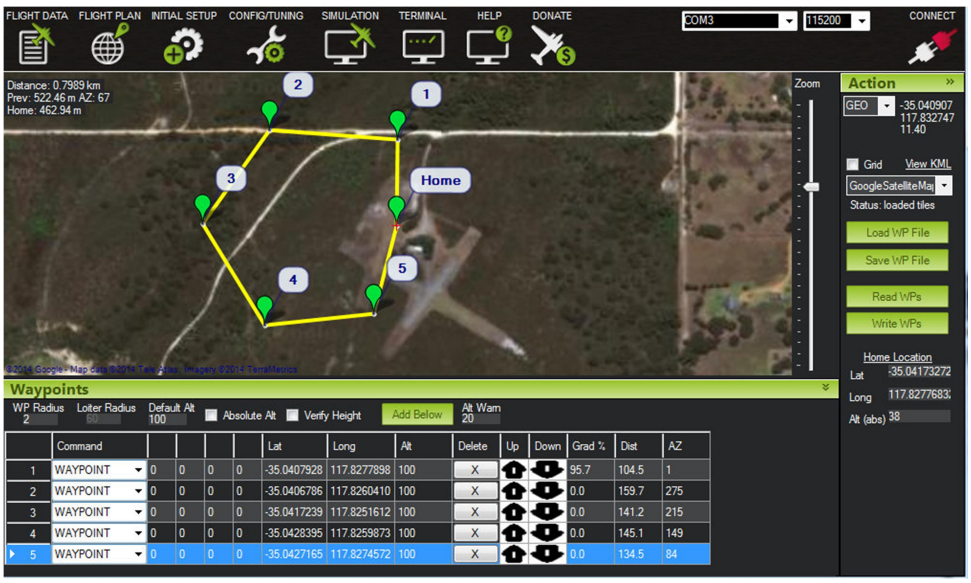

Figura 61: diseño de una misión con el software MissionPlanner

El desarrollo de un software propio para cargar misiones está justificado aún existiendo ya otros, ya que es un paso previo de cara a la programación de algoritmos que generen misiones de forma inteligente. Algunos desarrollos futuros en esta línea podrían ser:

- Generación automática de rutas para cubrir un área: el objetivo de este desarrollo sería poder cubrir un área determinada con un multirrotor que lleve a bordo algún tipo de sensor, como por ejemplo una cámara de vídeo. Los datos de entrada al programa serían una seríe de puntos geográficos, los cuales definirían el área a explorar, los puntos de despegue y de aterrizaje, y un parámetro que defina el ancho de pasillo deseado, es decir, la distancia entre las sucesivas trayectorias que el multirrotor realizará sobre el área. En el caso concreto de un sensor tipo cámara de vídeo, dicho ancho de pasillo sería función del ángulo de visión de la cámara y de la altura del vuelo. El programa generaría la sucesión de *waypoints* óptima (distancia recorrida menor) para llevar a cabo la exploración y los cargaría en el autopiloto.

- Desarrollo similar al anterior pero utilizando un número “n” de multirrotores. Este programa generaría "n" misiones distintas, una para cada multirrotor, de forma que en conjunto optimicen la distancia recorrida y cubran el área satisfactoriamente y en un tiempo menor del que emplearía un multirrotor solo.

El programa se ha realizado en Python 2, utilizando Dronekit para tener acceso de alto nivel al

protocolo MavLink. Fueron estudiadas distintas arquitecturas posibles para el desarrollo de Mission Loader. Dichas arquitecturas se describen a continuación:

- El programa se ejecuta en un PC, conectado al autopiloto mediante cable USB. El procedimiento sería el siguiente:
    - El autopiloto del multirrotor se conecta al ordenador mediante cable USB.
    - Se realiza la carga de la misión.
    - Se desconecta el cable USB. La misión está ya disponible y será ejecutada al activar el modo Auto.
- El diseño consta de dos programas distintos, uno que se ejecutará en un PC y el otro en la Raspberry Pi, a bordo del multirrotor. Se llamarán, respectivamente, programa 1 y programa 2. Ambos, PC y Raspberry, formarían parte de la misma red inalámbrica durante el proceso. El procedimiento en este caso sería el siguiente:
    - PC y Raspberry Pi se conectan a la misma red inalámbrica. Dicha red puede ser ajena a ellos (creada con un dispositivo externo) o puede ser generada por uno de los dos dispositivos, que actuaría como punto de acceso Wifi.
    - El operador usaría el programa 1, en para definir la misión y enviarla al multirrotor mediante algún protocolo de comunicación informático como TCP (*Transmision Control Protocol*) o UDP (*User Data Protocol*).
    - El Programa 2 recibiría la misión y la cargaría en el autopiloto.
    - La misión estaría ya disponible en el autopiloto y, por tanto, se ejecutaria al activar el modo Auto.
- Diseño con un único programa, el cual se ejecuta a bordo del multirrotor, en la Raspberry Pi. La Raspberry Pi y el PC deben estar, al igual que en el desarrollo anterior, conectados a la misma red inalámbrica. El procedimiento con esta arquitectura sería el siguiente:
    - PC y multirrotor (Raspberry Pi) se conectan a la misma red Wifi.
    - El usuario se conecta desde el PC a la Raspberry PI mediante un cliente de escritorio remoto.
    - A través de la conexión por escritorio remoto, se realiza la carga de la misión.

Se eligió la última arquitectura de las previamente descritas por considerarse la más práctica en función de las ventajas que se exponen a continuación:

- Permite realizar la carga de la misión sin conectar ningún cable desde el PC al multirrotor

- Únicamente implica el desarrollo de un *script,* el cual se ejecuta en la Raspberry Pi.
- En caso de configurar la Raspberry Pi como punto de acceso Wifi, cualquier operador podría conectarse a la red creada (introduciendo la contraseña establecida) y cargar una misión.
- Al existir clientes de escritorio remoto para Windows, Linux, Android y Mac, puede realizarse la carga de una misión prácticamente desde cualquier dispositivo.

4.3.1.2 Versiones e Interfaz de usuario

Se han llevado a cabo dos versiones del desarrollo Mission_Loader, el primero de ellos llamado *mission_scripty.py* y el segundo *mission_loader.py.* Ambos utilizan el mismo código para cargar la misión en la placa autopiloto. Sin embargo, presentan las siguientes diferencias:

- mission_script es un script de python que puede ser lanzado desde la consola de comandos de Linux. Al ejecutarlo, el script carga en la placa de autopiloto una misión de 5 waypoints que está definida en el propio código del script. Por tanto, para modificar la misión, hay que hacerlo directamente sobre el código fuente, en python.
- mission_loader es un script que, al ser ejecutado, abre una interfaz gráfica de usuario o GUI (Graphic User Interface) como la que se muestra en la Figura 62: mission_loader.

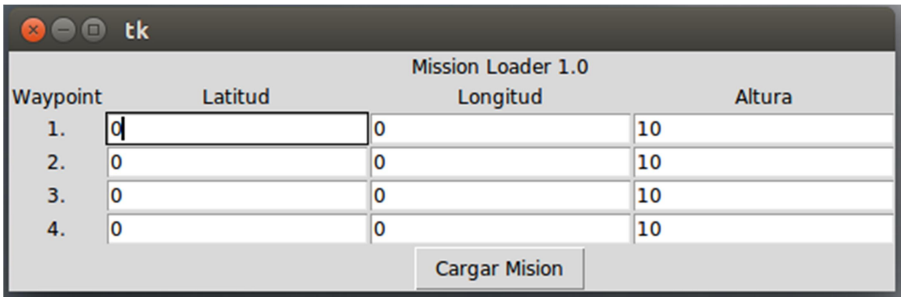

Figura 62: mission_loader

Contiene una serie de campos donde introducir las coordenadas (latitud, longitud y altura) de 4 waypoints, que conformarán la misión. El programa añadirá a la misión automáticamente la orden de despegue al inicio y de RTL (Return to Launch Site o Vuelta a casa) al final de la misión.

4.3.2 Pi2Pi Communication

Pi2Pi Communication es el nombre elegido para agrupar a aquellos desarrollos cuyo que tienen como fin cumplir con el tercero de los objetivos planteados para este proyecto: comunicación en vuelo entre varios multirrotores y compartición de algún tipo de información proveniente de los sensores embarcados. Mientras se produce este intercambio de datos, los multirrotores pueden estar volando de forma autónoma o controlados

por un piloto.

Se ha planteado un escenario en el que son dos los multirrotores que intervienen en la comunicación. Ambos multirroores se conocerán como Quad_1 y Quad_2 respectivamente. Quad_2 transmitirá a Quad_1 la información útil, y será Quad_1 el encargado de almacenarla para su posterior inspección.

Con objeto de simplificar el sistema, se ha considerado que la información útil serán las coordenadas GPS de Quad_2. Así pues, durante un vuelo, Quad_2 transmitirá a Quad_1 sus coordenadas GPS (latitud, longitud, altura) de forma continua, y Quad_1 almacenará esta información en un archivo de texto junto con sus propias coordenadas GPS. Esta forma de proceder permitirá:

- Validar el funcionamiento de Pi2Pi Communication
- Comprobar el alcance de la comunicación Wifi entre los dos multirrotores

Para implementar la conexión Wifi entre ambos multirrotores, uno de ellos actuará como punto de acceso Wifi mientras que el otro se conectará automáticamente a la red creada por el primero.

Para intercambiar la información se ha usado el protocolo TCP [16] (Transmision Control Protocol o Protocolo de Control de Transmisión), El propósito del protocolo TCP es garantizar un flujo de bytes confiable de extremo a extremo, siendo el protocolo el encargado de adaptarse dinámicamente a las propiedades de la red y manejar incidencias de distinto tipo.

Para establecer un servicio, el emisor y el receptor deben crear los puntos terminales de la conexión, llamados sockets. La dirección de un socket es la dirección IP del host y un número de 16 bits (puerta). Las conexiones de TCP son punto-a-punto y full dúplex. Lo concerniente a la gestión de la comunicación TCP se ha realizado a través del módulo *socket*, para python.

Para cada uno de los desarrollos realizados en esta línea (Pi2Pi Communication) se han realizado dos scripts complementarios: uno que actúa como servidor TCP y otro como cliente. Se ha cumplido el objetivo global del proyecto mediante una aproximación en tres pasos, cada uno de los cuales lleva asociado dos scripts. A continuación se describen dichos desarrollos:

- *Pi2Pi_Com_TCP_Tests_Client* y *Pi2Pi_Com_TCP_Tests_Server*: estos dos scripts implementan una comunicación TCP entre dos aplicaciones que se ejecutan en el mismo PC. Fueron desarrollados para realizar pruebas con la comunicación TCP la librería socket, dejando la problemática de implementación en las Raspberry Pi de lado.
- *Pi2Pi_Com_Tests_Quad1 y Pi2Pi_Com_Tests_Quad2*: estos scripts implementan una comunicación TCP entre dos Raspberry Pi que se encuentran conectadas a la misma red. Sin embargo, la información enviada es una cadena de caracteres invariante. El propósito de este desarrollo fue implementar una comunicación TCP funcional entre dos Raspberry Pi, aislando la problemática concerniente a obtener información de forma continua de las placas autopiloto.

- *Pi2Pi_Com_Quad1 y Pi2Pi_Com_Quad2*: se trata de los scripts finales. Implementan la comunicación TCP entre dos Raspberry Pi. Pi2Pi_Com_Quad2 toma de su autopiloto asociado las coordenadas GPS y las envía mediante TCP a la otra Raspberry Pi, en la que se ejecuta el script Pi2Pi_Com_Quad1. Dicho script se encarga de almacenar, en un archivo de texto, las coordenadas de ambos multirrotores, en un formato como el siguiente:

```
Drone 1: 37.914174,-4.7481549,182.11 Drone2: 37.9139583,-4.7486248,181.95
```

4.4 Aspectos prácticos del desarrollo software

En este subcapítulo se desarrollan algunos aspectos importantes de los desarrollos realizados, los cuales no han tenido cabida en los subcapítulos anteriores.

4.4.1 Instalación del Sistema Operativo en Raspberry Pi

Para empezar a utilizar las Raspberry Pi, el primer paso es instalar el sistema operativo. Las RPi carecen de disco duro, por tanto, el sistema operativo se instalará en una tarjeta de memoria del tipo micro SD.

El sistema operativo en su versión más reciente está disponible en la página web de Raspberry Pi Foundation, la organización que comercializa las placas y da soporte post-venta. De allí se descargó la imagen más reciente, la *2016-05-10-raspbian-jessie*. Se trata de una distribución de Linux optimizada para su uso en Raspberry Pi.

Una vez el archivo ha sido descargado, éste debe ser grabado en la tarjeta de memoria. Para ello se ha hecho uso del programa Win32IMager (Figura 63: programa Win32DiskImager). En este simple programa se seleccionan la unidad en la que va a grabarse la imagen (la correspondiente a la tarjeta microSD) y el archivo de imagen (extensión ISO) que va a grabarse y se pulsa el botón *Write*. En cuestión de minutos, la tarjeta SD está lista para ser utilizada en una Raspberry Pi.

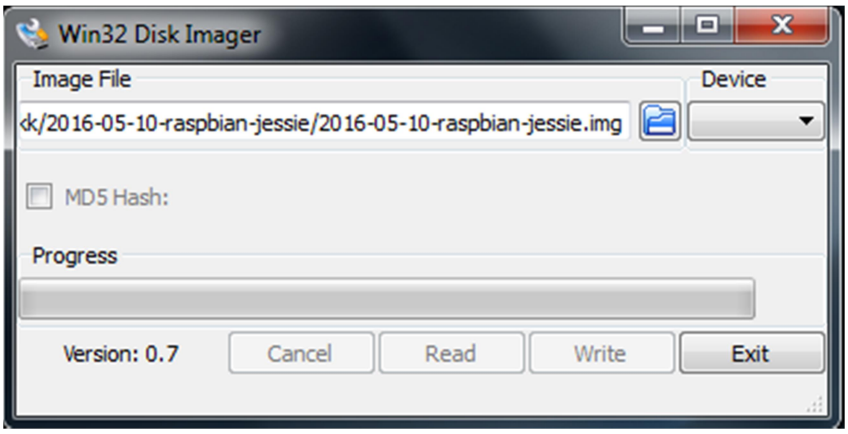

Figura 63: programa Win32DiskImager

4.4.2 Configuración inicial de Raspberry Pi

Una vez el sistema operativo ha sido instalado, es conveniente realizar una serie de acciones antes de empezar a utilizar la Raspberry Pi de forma continua. A continuación se describen dichas acciones y se explica cómo llevarlas a cabo.

4.4.2.1 Expandir Sistema de archivos

Esta acción sirve para que la totalidad del espacio de la tarjeta de memoria microSD en la que se ha grabado la imagen con el sistema operativo esté disponible para su uso. Esto se hace siguiendo, desde el escritorio de la RPi, la ruta *Menú, Preferencias, Configuración de Raspberry Pi* (Figura 64: ruta hasta *Configuración de Raspberry Pi*).

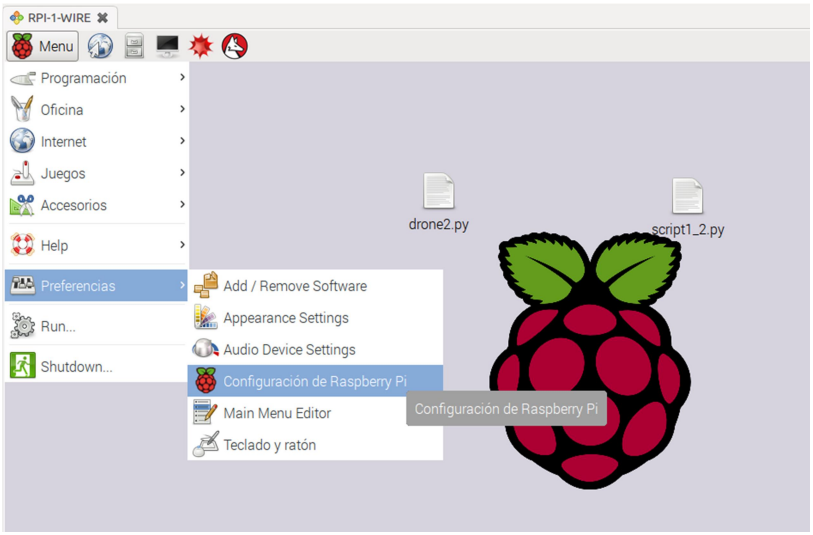

Figura 64: ruta hasta *Configuración de Raspberry Pi*

A continuación, en la pestaña Sistema, hay que hacer click sobre el botón *Expandir Sistema de archiv,* indicado en la Figura 65: botón de *Expandir Sistema de archivo.*

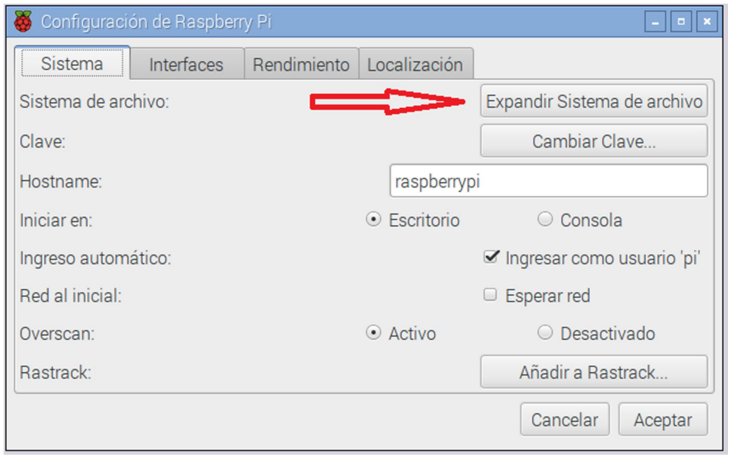

Figura 65: botón de *Expandir Sistema de archivo*

4.4.2.2 Elegir idioma

En la aplicación *Configuración de Raspberry Pi*, también puede elegirse el idioma del sistema. Dentro de la pestaña *Localización*, hay que pulsar el botón *Configurar Local* (Figura 66: selección de idioma en Raspberry Pi (I).

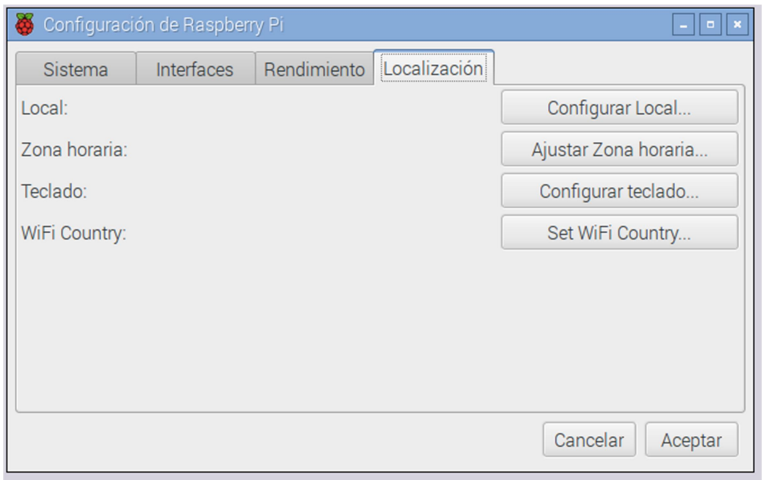

Figura 66: selección de idioma en Raspberry Pi (I)

Al pulsar el botón se abre una ventana en la que podemos elegir el idioma. En esta ventana (Figura 67: selección de idioma en Raspberry Pi (II) también puede elegirse el país donde se ubica la Raspberry Pi y el conjunto de caracteres deseado.

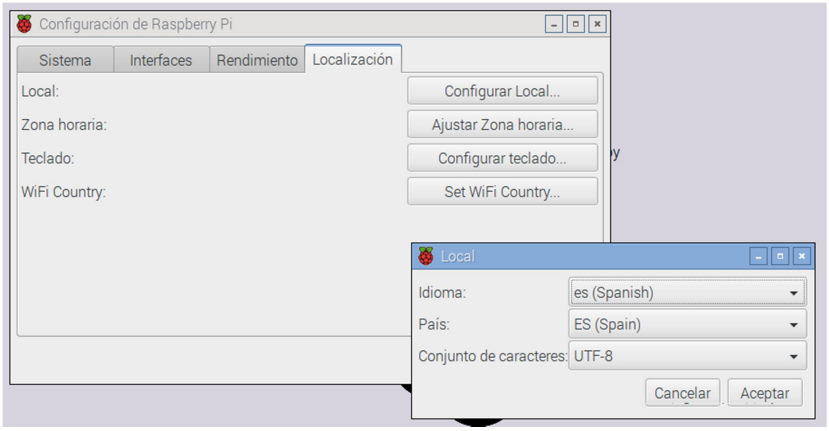

Figura 67: selección de idioma en Raspberry Pi (II)

4.4.2.3 Cambiar configuración del teclado

Por último, puede configurarse el teclado para que las teclas respondan adecuadamente. Esto se hace siguiendo la ruta *Menú, Preferencias, Teclado y ratón* (Figura 68: configuración del teclado (I).

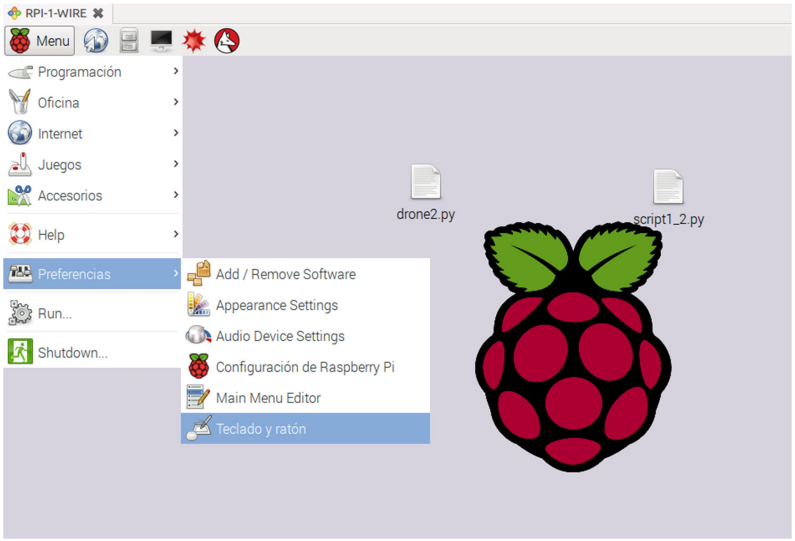

Figura 68: configuración del teclado (I)

En la pestaña *Teclado* podremos seleccionar alguns ajustes del teclado (retardo de repetición, intervalo de repetición) y, lo que es más importante, podremos elegir la distribución de teclas que se adapte al teclado físico del usuario (haciendo click en el botón Keyboard Layout, Figura 69: configuración del teclado (II)).

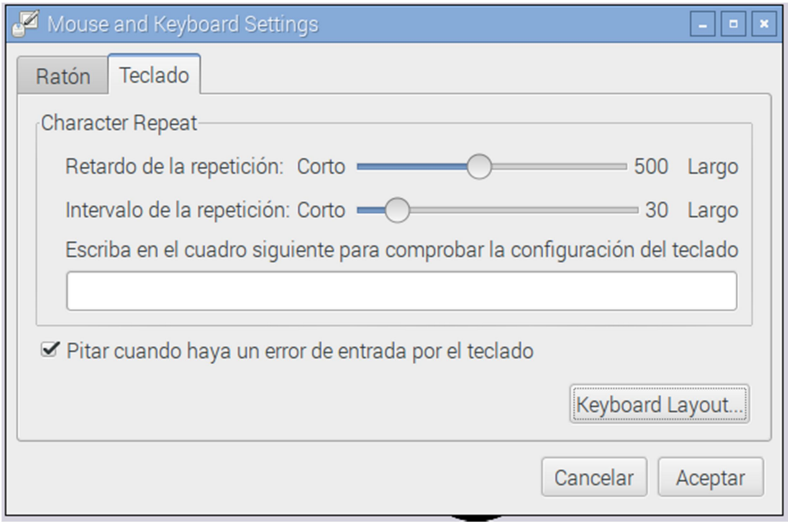

Figura 69: configuración del teclado (II)

4.4.2.4 Actualización del software instalado

Esta operación consiste en actualizar el software que viene instalado con el sistema operativo a su versión más reciente. Para realizar esta operación, se debe abrir un terminal o consola de comandos y ejecutar los siguientes comandos:

```
sudo apt-get update
sudo apt-get upgrade
```

El primero de los comandos actualiza los repositorios, es decir, comprueba si hay actualizaciones disponibles sobre el software ya instalado. Cuando este comando termina, se ejecuta el segundo y se instalarán las actualizaciones disponibles. El proceso puede tardar unos minutos.

4.4.3 Instalación software

Una vez instalado el sistema operativo y concluída la configuración inicial, la Raspberry Pi está lista para instalar en ella todo el software que posteriormente será útil para acometer los desarrollos planteados en este proyecto.

En los próximos subcapítulos se explica cómo instalar dichos programas, operaciones en que será necesario disponer de conexión a internet en la Raspberry Pi.

4.4.3.1 Python-pip y Python-dev

El intérprete del lenguaje Python viene instalado por defecto con el sistema operativo y, con la actualización de software previa, nos aseguramos de que se encuentra en su versión más reciente. Sin embargo, para poder trabajar con Dronekit, será necesario instalar dos paquetes más. A continuación se muestran los comandos para instalar ambos paquetes de software:

```
sudo apt-get install python-pip
sudo apt-get install python-dev
```

4.4.3.2 Dronekit

Como se explica en el capítulo 4.2.4, Dronekit es la API que se ha utilizado para desarrollar las aplicaciones que se comunicarán con la placa de autopiloto. Para instalar dicha API en la Raspberry, se introducen en el terminal los siguientes comandos:

```
sudo pip install dronekit
sudo pip install dronekit-sitl
```

El primero de los comandos instala la API y el segundo instala un simulador de vehículos autónomos. Su utilidad es poder llevar a cabo pruebas sin necesidad de realizar un vuelo de verdad. Este simulador no ha sido utilizado en este proyecto, pero podría resultar muy interesante de cara a acelerar el desarrollo de aplicaciones.

4.4.3.3 Cliente de escritorio remoto

La Raspberry Pi debe tener instalado un cliente de escritorio remote para poder controlarla remotamente desde otro dispositivo mediante el protocolo RDP o *Remote Desktop Protocol*. Para instalar el cliente de Linux, escribimos en el terminal el siguiente comando:

```
sudo apt-get install xrdp
```

4.4.4 Creación de programas con Python

Los desarrollos software del presente proyecto han sido realizados en Python. La instalación del intérprete de Python en una Raspberry con acceso a internet es muy sencillo y se hizo en 4.4.3. A la hora de hacer un programa en Python para Raspberry Pi, existen distintas formas de hacerlo, que se explican a continuación:

- Realizar los programas directamente en la Raspberry Pi.
- Realizar los programas en la Raspberry Pi, sin conectar esta a teclado y pantalla. En este caso se hará uso de algunas de las alternativas para controlar las rapsberry pi remotamente que se describen en 4.4.5.
- Realizar los programas en otro ordenador. Una alternativa interesante (la mayoritariamente utilizada en este proyecto para realizar trabajos en los que el hardware no fuera importante) es hacer los programas en un PC y posteriormente llevar los archivos a la Raspberry a través de una memoria externa, por ejemplo.

4.4.5 Control remoto de Raspberry Pi

La opción más convencional para trabajar con una Raspberry Pi consiste en conectarla a un monitor a

través de un cable HDMI asi como a un teclado y un ratón mediante cables USB. De este modo se puede utilizar como si se tratase de un PC convencional. Sin embargo, esta opción en ocasiones resulta poco práctica por las razones que se describen a continuación:

- No siempre se dispone de un monitor HDMI, teclado y ratón para poder trabajar con una Raspberry.
- Si la Raspberry Pi se encuentra instalada en algún lugar concreto (sobre un multirrotor, por ejemplo), resulta incómodo y a veces incluso imposible conectarle los periféricos antes mencionados.

La alternativa más adecuada consiste en controlarla remotamente desde un ordenador, para lo cual es necesario que el ordenador y la Raspberry estén conectados a la misma red, ya sea mediante conexión inalámbrica (conectados a la misma red Wifi) como cableada (conectando un cable de red entre la Raspberry y el ordenador).

4.4.5.1 Escritorio remoto

El protocolo de escritorio remoto (RDP o Remote Desktop Protocol) es un protocolo desarrollado por Microsoft, basado en T-120, y que permite acceder a un escritorio desde una ubicación distante. El proceso consiste en codificar en formato RDP la información generada por el equipo que toma el rol de servidor y mostrarla en aquel que juega el rol de cliente.

RDP permite canales virtuales independientes para transportar datos de presentación, comunicación de dispositivos serie, información de licencias y datos cifrados (pulsaciones de teclado, actividad del mouse, etc). Al tratarse de una extensión del núcleo del protocolo T.Share RDP, se conservan capacidades como por ejemplo los elementos arquitectónicos necesarios para soportar sesiones multipunto (con varios participantes). Esta particularidad permite que los datos de una aplicación se entreguen en varias ubicaciones sin tener que enviar los mismos datos para cada sesión de forma individual.

Para controlar la Raspberry Pi mediante este protocolo se ha instalado el cliente xrdp en la misma, como se verá más adelante en el punto 4.4.3.3. Se ha utilizado un PC con Ubuntu para controlar remotamente la Raspberry. La distribución Ubuntu 14.04 LTS incluye por defecto un cliente de escritorio remoto llamado *Remmina*.

4.4.5.2 SSH

SSH (*Secure Shell*) [17] es un protocolo seguro de comunicaciones entre dos sistemas mediante una arquitectura cliente/servidor que permite a un usuario conectarse a un *host* remotamente. A diferencia de otros protocolos de comunicación remota, SSH encripta la sesión de conexión y está diseñado para reemplazar otros métodos menos seguros para registrarse remotamente en otro sistema a través de consola de comandos, como

por ejemplo *telnet* o *rsh.*

El protocol SSH propociona los siguientes tipos de proteccón:

- Después de la conexión inicial el cliente puede verificar que sigue conectado al mismo servidor al que se conecto anteriormente.
- El cliente transmite su información de autenticación al servidor una encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos se transfieren mediante encriptación de 128 bits.

La principal diferencia entre utilizer RDP o SSH para controlar remotamente la Raspberry Pi es que mediante RDP se puede arrancar el entorno gráfico y utilizar aplicaciones de escritorio, mientras que mediante SSH únicamente puede controlarse la RPi a través de una consola de comandos.

4.4.6 Configurar Raspberry Pi como punto de acceso Wifi

Durante la realización de los desarrollos de este proyecto se utilizaron principalmente dos RPis. La configuración utilizada tanto durante el desarrollo como durante las pruebas del desarrollo Pi2Pi Communication es la siguiente:

- Una RPi, denominada RPi_1, actúa como punto de acceso Wifi. Dicha RPi, haciendo uso del dongle Wifi a la que está conectada, crea una red Wifi llamada DroneNet.
- La otra Raspberry Pi, llamada RPi_2, se encuentra configurada para conectarse automáticamente a DroneNet cuando DroneNet esté disponible, manteniendo una IP fija.

Con esta configuración, una vez alimentadas sendas RPis se establece la red DroneNet y, por tanto, RPi_2 se conecta a RPi_1. A continuación, puede utilizarse un PC para conectarse a DroneNet y controlar remotamente sendas RPis, sin tener que establecer ninguna conexión física entre los 3 elementos. Para configurar una Raspberry Pi como punto de acceso se han seguido los siguientes pasos [18]:

1. En primer lugar hay que instalar los paquetes software *hostapd* y *udhcpd*.

   ```
   sudo apt-get install hostapd udhcpd zd1211-firmware
   ```

   hostapd es un servidor de punto de acceso con soporte para autenticación IEEE 802.11 y IEE 802.1X/WPA/WPA2/EAP y udhcpd es un servidor DHCP (*Dynamic Host Configuration Protocol* o Protocolo de configuración dinámica de host) ligero y habitualmente utilizado en sistemas embebidos. Por ultimo, *zd1211* es un driver utilizado por *hostapd.*

2. Configurar el servidor DHCP. Es necesario editar el archive que se ubica en la ruta /etc/udhcpd.conf . Para editarlo, se introduce en la consola la siguiente instrucción:

   ```
   sudo vi /etc/udhcpd.conf
   ```

A continuación se muestra el contenido que hay que introducir en el archivo:

```
# The start and end of the DHCP lease block

start         192.168.0.20
end           192.168.0.254

# The wireless interface used by udhcpd

Interface     wlan0

# If remaining is true (default), udhcpd will store the time
# remaining for eache lease in the udhcpd leases file. This is
# for embedded systems that cannot keep time between reboots.

 Remaining     yes

# The location of DHCP lease file

Lease_file    /var/lib/misc/udhcpd.leases

# The location of the pid file

Pidfile       /var/run/udhcpd.pid

# DNS servers that connected devices will use. Use Google DNS.

Opt   dns     8.8.8.8 8.8.4.4

# The IP address of the access point

Opt   router  192.168.0.1
Opt   subnet  255.255.255.0
Opt   domain  local

# 10 days of lease period
Opt   lease 864000

# Optionally specify static lease(s)

# static_lease   00:51:AF:05:B0:05  192.168.0.100
# static_lease   00:51:AF:00:E1:02  192.168.0.110
```

3. Crear un archivo vacío para lease DHCP, el cual udhcpd rellenará automáticamente a continuación.

```
sudo touch /var/lib/misc/udhcpd.leases
```

4. Habilitar udhcpd permanente. Para ello hay que abrir el archive /etc/default/udhcpd :

```
sudo vi /etc/default/udhcpd
```

A continuación, hay que comentar (añadir el character '#' al principio) la siguiente línea:

```
#DHCPD_ENABLED=”no”
```

5. Configurar udhcpd para que arranque automáticamente al iniciar el sistema. Se hace introduciendo el siguiente comando por consola:

```
Sudo update-rc.d udhcpd enable
```

6. Editar el archivo /etc/network/interfaces para asignar una dirección IP estática a la interfaz wlan0. Dicha dirección IP debe ser la misma que la definida previamente en el archivo /etc/udhcpd.conf Para abrir el archivo introducimos por consola:

```
Sudo vi /etc/network/interfaces
```

A continuación modificamos el contenido del archivo:

```
Auto lo

Iface lo inet loopback
Iface eth0 inet dhcp

Iface wlan0 inet static
Address 192.168.0.1
Netmask 255.255.255.0
```

7. Configurar el servidor HostAPD. Creamos el archivo de configuración introduciendo la siguiente línea por consola:

```
Sudo vi /etc/hostapd/hostapd.conf
```

Editamos el archivo, con el siguiente contenido:

```
# interface used by Access point
Interface = wlan0

# firmware driver
Driver = nl80211

# access point SSID
Ssid = rpiap

# operation mode(a=IEEE802.11a,b=IEEE802.11b,g=IEEE802.11g)
Hw_mode = g

# access point channel
Channel = 6

Macaddr_acl = 0
Auth_algs = 1
Ignore_broadcast_ssid = 0

# key management algorithm
Wpa_key_mgmt = WPA-PSK
Wpa_passphrase = mypasscode
Wpa = 2
```

```
#set ciphers
Wpa_pairwise = TKIP
Rsn_pairwise = CCMP
```

8. Editar el archivo /etc/default/hostapd para incluir una referencia al archivo de configuración anterior. Abrir el archivo con la instrucción:

```
Sudo vi /etc/default/hostapd
```

Incluir la siguiente línea en el archivo:

```
DAEMON_CONF = "/etc/hostapd/hostapd.conf"
```

9. Confifurar el servidor hostapd para que arranque automáticamente al iniciar la Raspberry Pi:

```
Sudo update-rc.d hostapd enable
```

10. Configurar direccionamiento IP NAT (Network Adress Translation o tradución de direcciones de red) para que el punto de acceso pueda comunicarse con redes externas en nombre de todos los dispositivos conectados. En primer lugar, para permitir el direccionamiento, introducimos la siguiente línea por consola:

```
Sudo vi /etc/sysctl.conf
```

En el archivo que se abrirá, escribimos la siguiente línea:

```
Net.ipv4.ip_forward=1
```

A continuación cerramos el archivo anterior e introducimos por consola las siguientes líneas:

```
Sudo iptables –t nat –A POSTROUTING –o eth0 –j MASQUERADE

Sudo iptables –A FORWARD –i eth0 –o wlan0 –m state – state
RELATED,ESTABLISHED –j ACCEPT

Sudo iptables –A FORWARD –i wlan0 –o eth0 –j ACCEPT

Sudo sh –c "iptables-save > /etc/iptables.ap"
```

Finalmente editamos el archivo /etc/network/interfaces y añadimos la siguiente línea al final del archivo:

```
Up iptables-restore < /etc/iptables.ipv4.nat
```

El archivo presentará el siguiente aspecto:

```
Auto lo

Iface lo inet loopback
Iface eth0 inet dhcp
```

```
Iface wlan0 inet static
Address 192.168.0.1
Netmask 255.255.255.0

Up iptables-restore < /etc/iptables.ap
```

11. Finalmente, se debe reiniciar la Raspberry Pi para que empiece a funcionar como punto de acceso Wi-Fi.

5 PRUEBAS DE VUELO

Para castigarme por mi desacato a la autoridad, el destino me hizo a mí mismo autoridad.

- Albert Einstein –

El presente capítulo expone las distintas pruebas de vuelo que han sido realizadas con los multirrotores, tanto en su configuración con equipamiento básico, como con el equipamiento extendido. Las pruebas con equipamiento básico tuvieron como objetivo comprobar el correcto funcionamiento y configuración de los multirrotores así como la sintonización de las distintas variables de control. Por otra parte, las pruebas con el equipamiento extendido sirvieron para probar los desarrollos realizados.

5.1 Primeros vuelos

Una vez el montaje del primer multirrotor estuvo concluido, se realizaron las primeras pruebas. Inmediatamente se apreciaron unas intensas vibraciones en los brazos, las cuales desaconsejaban totalmente continuar volando. Dichas vibraciones se acentuaban a unas determinadas revoluciones de los motores, produciéndose un fenómeno de resonancia. El estudio de unas grabaciones realizadas a cámara lenta durante las primeras pruebas permitió deducir las siguientes causas para dicho comportamiento:

- El modelo de chasis, teniendo en cuenta la potencia de los motores y el peso del conjunto, no es lo suficientemente rígido. Los brazos del chasis experimentan una flexión excesiva durante el vuelo, incluso realizando un vuelo estacionario.
- Las hélices presentan desequilibrios, problema que combinado con el anterior provocan un vuelo muy errático, no recomendable.

El problema fue solucionado equilibrando las hélices de carbono. Fueron desmontadas de los motores y colocadas en un equlibrador de hélices (Figura 70: equilibrador de hélices y Figura 71: equilibrador de hélices (detalle)), capaz de detectar pequeños desequilibrios. Para igualar los pesos se utilizaron dos métodos: lijar ligeramente el borde de salida de la pala más pesada y colocar pequeñas cintas adhesivas en la cara más ligera. Ambos han demostrado ser igualmente efectivos.

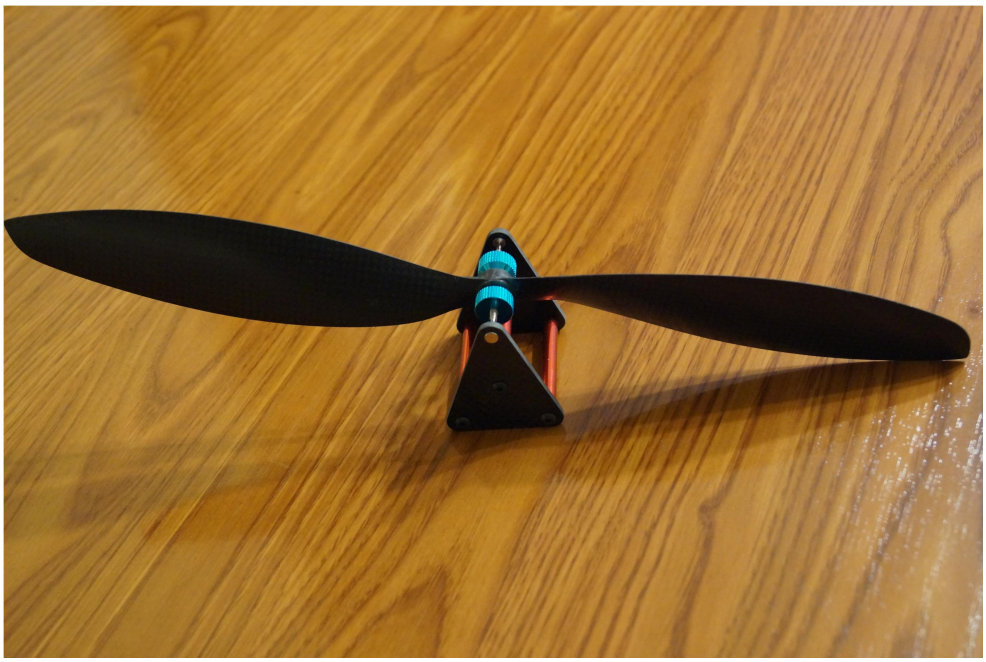

Figura 70: equilibrador de hélices

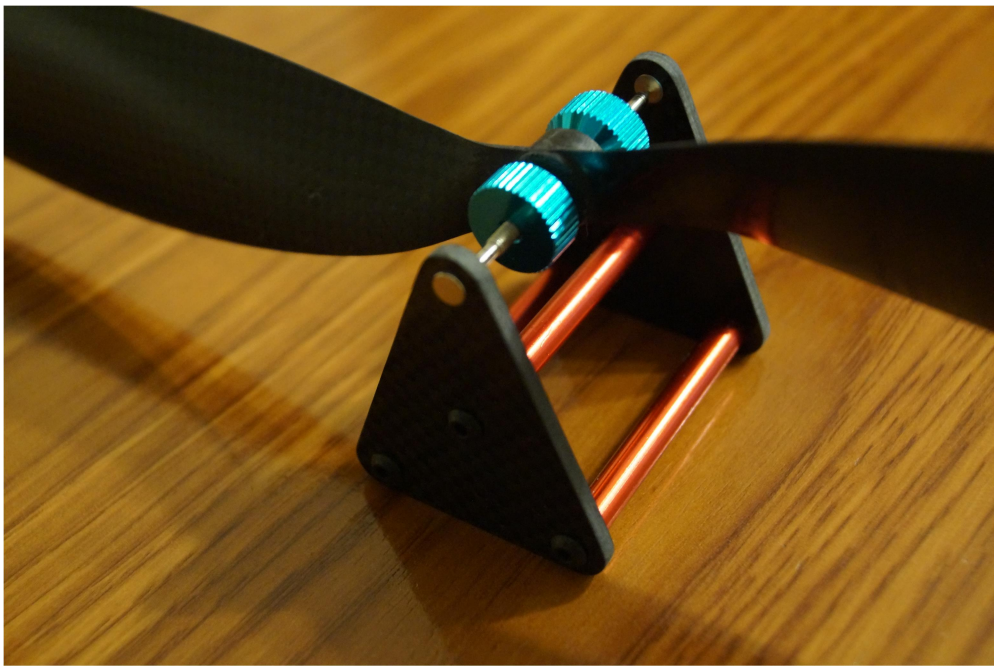

Figura 71: equilibrador de hélices (detalle)

Una vez resuelto el problema anteriormente descrito, se procedió a sintonizar el control. Como el autopiloto utilizado es compatible con cualquier multirrotor (incluso otros vehículos), es necesario un proceso de sintonización del control éste se adapte al comportamiento del multirrotor en que está instalado.

Se probó en primer lugar a utilizar el modo de vuelo de sintonización de la placa Ardupilot. Al activar este modo, una vez que el multirrotor despega, si el piloto deja los mandos libres (sin comandar nada), el multirrotor efectúa movimientos automáticamente y, en base a su propio comportamiento, va ajustando las variables de control.

Después de realizar un vuelo de auto-sintonización, el comportamiento del multirrotor seguía sin ser totalmente satisfactorio. Por esa razón, se realizaron 2 vuelos más en los que las variables de control fueron ajustadas manualmente hasta que el comportamiento del multirrotor fue aceptable.

5.2 Pruebas de los desarrollos

En este subcapítulo se narrarán las distintas pruebas efectuadas con los desarrollos software y se evaluará si son consideradas satisfactorias o no.

5.2.1 Pruebas sobre Mission_Loader

Si bien durante la realización de este desarrollo fueron llevadas a cabo pruebas progresivas, una vez estuvo finalizado se llevó a cabo una prueba final. La filosofía que se ha seguido durante el proyecto es la de realizar el menor número posible de vuelos con los multirrotores, con objeto de minimizar el riesgo de accidente. Por este motivo y dado que no era necesario realizar un vuelo para validar este desarrollo, la prueba

se realizó en tierra.

La prueba definitiva tenía que demostrar que el programa Mission_Loader podia cargar en la placa autopiloto los waypoints definidos por el usuario mediante los parámetros de longitud, latitud y altura de forma satisfactoria. Constó de los siguientes pasos:

1. Primera fase: cargar mission
    a. Se realizó un montaje que incluía una Raspberry Pi, modulo Wifi, placa de autopiloto y una fuente de alimentación.
    b. Se conectó dicha Raspberry Pi así como un PC con Ubuntu a la misma red inalámbrica, mediante Wifi.
    c. Desde el PC con Ubuntu se accedió remotamente a la Raspberry Pi utilizando el protocolo RDP (Remote Desktop Protocol o Protocolo de Escritorio Remoto).
    d. A través del enlace remote, se ejecutó el programa Mission_Loader y se procedió a cargar una misión en la placa de autopiloto
2. Segunda fase: verificar misión correctamente cargada
    a. Se tomó la misma placa de autopiloto y se conectó mediante USB a un PC cuyo sistema operativo era Windows 7.
    b. Mediante la aplicación MissionPlanner se descargó la misión de la placa de autopiloto y se visualizó en el ordenador.

El resultado fue satisfactorio pues se pudo comprobar que la misión se cargó correctamente y pudo ser visualizada posteriormente con MissionPlanner sobre un mapa.

5.2.2 Pruebas de Pi2Pi Communication

El objetivo del desarrollo Pi2Pi_Communication consistía en intercambiar información entre dos multirrotores en vuelo. Como se explicó en el capítulo dedicado a exponer lo referente a este desarrollo, la información que se decidió compartir fue la posición GPS de uno de los multirrotores.

Siguiendo la línea de las pruebas del desarrollo Mission_Loader , se ha hecho todo lo possible por realizar una prueba con el menor riesgo e incertidumbre posible. La forma de conseguirlo ha consistido es reducir el número de multirrotores durante la prueba de dos a uno. El denominado Quad_1 consistió en un multirrotor real que efectuó un vuelo manual. Sin embargo, el Quad_2 consistió en un montaje realizado en tierra, el cual incluyó una Raspberry Pi, autopiloto APM y su modulo GPS, todo ello alimentado desde la batería de un coche (Figura 72: autopiloto APM y módulo GPS, instalados en tierra. y Figura 73: montaje en tierra que incluye Raspberry Pi,).

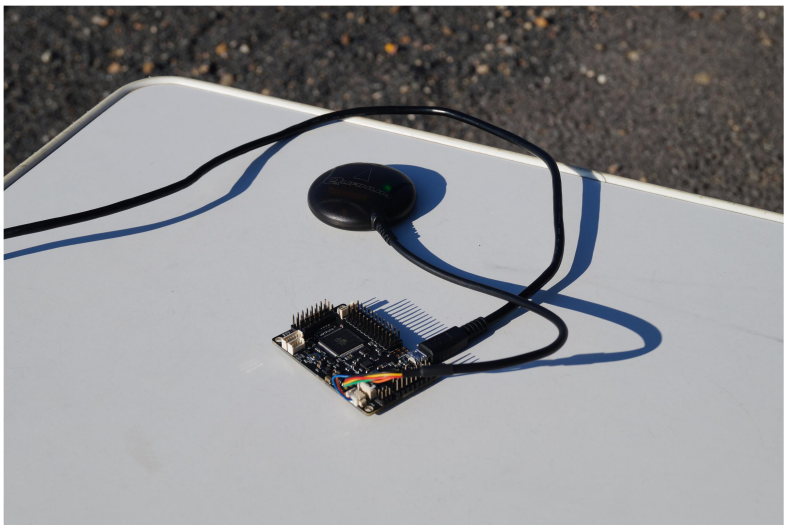

Figura 72: autopiloto APM y módulo GPS, instalados en tierra.

Figura 73: montaje en tierra que incluye Raspberry Pi,
módulo Wi-Fi, placa APM y módulo GPS.

Los objetivos de la prueba de este desarrollo fueron los siguientes:

- Validar el funcionamiento de Pi2Pi Communication: comprobar que se produce la conexión TCP entre las dos Raspberry Pi, y que se transmite la información y se almacena adecuadamente en un archivo de texto.
- Comprobar el alcance de la comunicación Wifi entre los dos multirrotores: durante la prueba, el multirrotor comenzó su vuelo a una distancia elevada del montaje en tierra, acercándose cada vez más hasta un punto cercano (unos metros de distancia) y posteriormente se alejó de nuevo. Esto permitió evaluar el alcance de la comunicación Wifi y valorar su idoneidad o no para este

tipo de desarrollos.

Figura 74: puesta en marcha del multirrotor

En la Figura 74: puesta en marcha del multirrotor puede verse la puesta en marcha del multirrotor. Una vez alimentadas la Raspberry Pi embarcada y la que permanece en tierra, se iniciaron los scripts a través de un ordenador portátil mediante escritorio remoto Figura 75: arranque de los scripts mediante RDP.

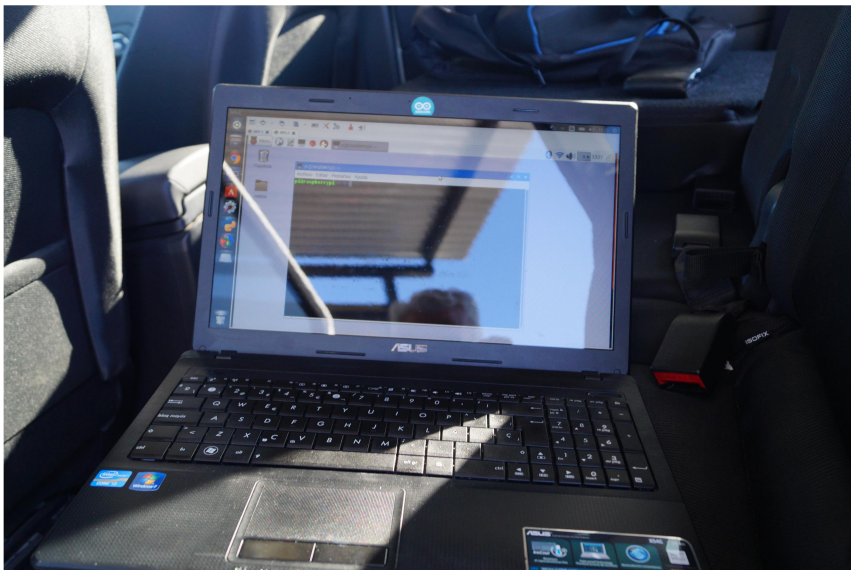

Figura 75: arranque de los scripts mediante RDP.

Las figuras Figura 76: vuelo del multirrotor durante la prueba (I), y Figura 77: vuelo del multirrotor durante la prueba (II). muestran el vuelo del multirrotor durante la prueba.

Figura 76: vuelo del multirrotor durante la prueba (I),

Figura 77: vuelo del multirrotor durante la prueba (II).

A continuación, la Tabla 7: coordenadas de los puntos representativos de la prueba de Pi2Pi Communication muestra los puntos más significativos del vuelo de prueba de Pi2Pi Communication: el punto

de despegue, de entrada en la zona Wi-Fi y de salida de la zona Wi-Fi. También se muestra el punto en que se encontraba la Rasperry Pi en tierra.

	Latitud	**Longitud**
Despegue	37.9137373	-4.7492328
Entrada en zona Wi-Fi	37.9139296	-4.7487767
Entrada en zona Wi-Fi	37.9140524	-4.7486246
RPi en tierra	37.9139587	-4.7486246

Tabla 7: coordenadas de los puntos representativos de la prueba de Pi2Pi Communication

Finalmente, la Figura 78: resumen del vuelo de prueba de Pi2Pi Communication muestra los puntos representativos sobre una imagen por satellite del lugar. Si bien el desempeño del software desarrollado fue el esperado, los módulos Wi-Fi han demostrado tener un alcance insuficiente para su uso en multirrotores, el cual no superó los 14 metros durante las pruebas.

Figura 78: resumen del vuelo de prueba de Pi2Pi Communication

6 CONCLUSIONES Y DESARROLLOS FUTUROS

Vivimos en una sociedad profundamente dependiente de la ciencia y la tecnología y en la que nadie sabe nada de estos temas. Ello constituye una fórmula segura para el desastre.

- Carl Sagan –

El presente capítulo es el desenlace de este proyecto y pretende ser una síntesis de los aspectos más importantes del mismo. Adicionalmente se expondrán las principales conclusiones a las que el autor ha llegado durante su redacción, se presentarán las posibles mejoras aplicables a los desarrollos realizados y por último se enumerará una serie de posibles desarrollos futuros, continuación lógica de los aquí descritos, y que al autor le gustaría poder ver algún día en funcionamiento.

6.1 Resumen general del proyecto

El proyecto realizado es una primera aproximación a la creación de redes de comunicaciones aéreas basadas en vehículos aéreos autónomos de tipo multirrotor. Se ha tratado una temática muy diversa en el marco teórico, la cual ha incluído una extensa introducción a las aeronaves no tripuladas, sus tipologías, características y la nomenclatura asociada. Posteriormente se profundizó en las aeronaves de tipo multirrotor, desde un punto de vista físico y de las exigencias de control. Por último se trató la temática de los sitemas de comunicaciones involucrados en la operación de sistemas aeronaves no tripuladas.

Una vez expuesto el núcleo del marco teórico el capítulo 3 sirvió para, desde un punto de vista práctico, exponer cada uno de los sistemas que componen un multirrotor haciendo hincapié en los productos comerciales concretos que se han utilizado en este proyecto. La segunda parte de dicho capítulo es una guía (y ha sido utilizado como tal) de montaje y configuración de multirrotores. El grueso de este capítulo fue redactado durante el montaje del primer multirrotor. Posteriormente, fue consultado y revisado durante el montaje de los siguientes.

Los desarrollos prácticos, realizados en lenguaje Python, han servido para comprender la forma de utilizar la combinación hardware Raspberry Pi + Autopiloto APM y de comunicar ambos dispositivos a través del protocolo MavLink, gestionado mediante la API Dronekit.

6.2 Conclusiones sobre el trabajo realizado

La primera acción que se llevó a cabo para acometer este proyecto fue la selección de los materiales que constituirían los multirrotores. Posteriormente se acometió su montaje y, por último, los desarrollos software.

Las conclusiones del autor respecto de la selección de materiales y el montaje de los multirrotores se exponen a continuación:

- La selección de los materiales empleados puede considerarse muy satisfactoria, pues los multirrotores montados cumplen adecuadamente con las características deseadas en lo que respecta a tiempo de vuelo, capacidad de carga y aptitudes de vuelo.
- La única excepción a lo anteriormente comentado la constituye la selección de los chasis, los cuales sin duda carecen de la rigidez deseada. Como se explicó en el capítulo 5.1, el reforzamiento de los mismos o su sustitución deben ser una prioridad para aquellos que tengan la intención de utilizar estos multirrotores para algún tipo de trabajo.

Respecto de los desarrollos software realizados, pueden ser considerados un éxito pues cumplen con los objetivos planteados para este proyecto y que fueron descritos en el capítulo primero. Sin embargo, el alcance conseguido con los módulos wifi utilizados solo puede considerarse útil para realizar experimentos a nivel académico, siendo claramente insuficiente para su utilización en redes de comunicaciones con ambiciones

mayores.

6.3 Desarrollos futuros

Los desarrollos futuros se dividen en dos ramas: desarrollos hardware y software. El futuro de este proyecto en lo que respecta al hardware consiste, principalmente, en el refuerzo o sustitución de los chasis de los multirrotores y en cambiar los módulos Wifi utilizados para las comunicaciones por unos dispositivos de mayor potencia y/o antenas direccionales que ofrezcan mayor ganancia. Otra opción interesante sería ampliar el rango de las comunicaciones haciendo uso de la red GSM o comunicaciones satelitales.

En cuanto a los desarrollos software, a continuación se muestran las mejoras planteadas de las aplicaciones realizadas, así como una serie de posibles aplicaciones nuevas:

- Mission_Loader 2.0. Las siguients características serían deseables en una nueva versión del desarrollo Mission_Planner:
    - Posibilidad de elegir un número de waypoints diferente a 5.
    - Selección de waypoints sobre un mapa.
    - Posibilidad de seleccionar un área a cubrir, para que Mission_Planner genere la misión automáticamente.
    - Poder realizar el punto anterior con más de un multirrotor, para cubrir áreas en un tiempo menor.
- Mission Analizer: sería una aplicación para analizar un vuelo que ha sido realizado mediante Mission_Loader. Se ejecutaría durante el vuelo, grabando la información proveniente de los sensores junto con la posición actual en ese momento. Después del vuelo su función sería mostrar, sobre un mapa de la zona sobrevolada, la información captada por los sensores embarcados en forma de colores. Una posible aplicación sería realizar un vuelo sobre un área en la que existe un contaminante atmosférico de algún tipo y posteriormente, visualizando el mapa de colores, poder ver cuáles son las zonas de mayor concentración.
- GCS multivehículo: se trata de una aplicación tipo GCS para ser ejecutada en un ordenador. Su propósito sería realizar el seguimiento de los vuelos de varios multirrotores al mismo tiempo. Esta aplicación podría permitir visualizar los parámetros del vuelo en tiempo real, así como comandar órdenes a cada uno de los multirrotores por separado y por tanto permitiría a un único operador controlar las operaciones de varios multirrotores al mismo tiempo.
- Red de sensores: se trata de un desarrollo en el cual los multirrotores integrarían una red de sensores aérea. En el caso de implementarse dicha red mediante tecnología Wi-Fi, cada multirrotor (o nodo) debería llevar embarcados dos módulos, de forma que uno actúe como

punto de acceso y permita que otros nodos puedan conectarse a él, mientras el otro módulo es utilizado para conectarse a las redes creadas por los otros nodos (Figura 79: esquema básico de una red de sensores).

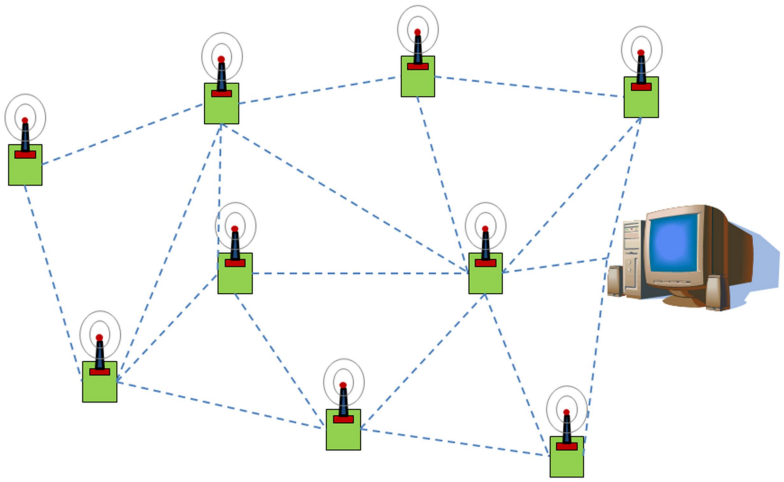

Figura 79: esquema básico de una red de sensores

Esta arquitectura permitiría extender el radio de acción, pues utilizando tantos nodos de respaldo como sea necesario, un multirrotor podría alejarse grandes distancias del punto de partida, hasta el punto en que el elemento limitante sea su propia autonomía energética.

Esta red aérea sería susceptible de implementar sensores de cualquier tipo para proveer cualquier tipo de servicio.

REFERENCIAS

[1] **R.K. Barnhart, S.B. Hottman, D. M. Marshall, E. Shappee**, "Chapter 1: History." in *Introduction to Unmanned Aircraft Systems*, R.K. Barnhart. Ed. Taylor & Francis Group LLC, 2012

[2] **J. Gundlach**, "Unmanned Aircraft Categories." In *Designing Unmanned Aircraft Systems: A Comprehensive Approach*, J.A. Schetz. Ed. AIAA Education Series, 2011

[3] **R. Austin**, "Introduction to Unmanned Aircraft Systems (UAS)" in *Unmaned Aircraft Systems, UAVs Design, Development and Deployment,* Ed. John Wiley & Sons, 2010

[4] **G. H. Elkaim, F.A.P. Lie, D. Gebre-Egziabher**, *Principles of Guidance, Navigation and Control of UAVs,* 2012

[5] **T. Luukkonen**, "Modelling and control of quadcopter", Proyecto independiente en matemáticas aplicadas, Aalto University, Espoo, Finland, 2011

[6] **A. Gibiansky**, Quadcopter Dynamics and Simulation [online], Disponible en http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/

[7] **Arducopter PID Loops for Stabilize, ACRO and ALT_HOLD.** Foro de la comunidad DIY Drones [online]. Disponible en http://diydrones.com/profiles/blogs/arducopter-2-9-pid-loops-for-stabilize-acro-and-alt-hold

[8] **S. Morgenthaler, T. Braun, Z. Zhao, T. Staub, M. Anwander**, *UAVNet: A Mobile Wireless Mesh Network Using Unmanned Aerial Vehicles , Bern University, Switzerland*

[9] **Turnigy Power Systems**, Página web del fabricante [online]. Disponible en http://www.turnigy.com

[10] **Ardupilot Copter**. Documentación oficial de Ardupilot [online], Disponible en *http://www.ardupilot.org/copter*

[11] **Raspberry Pi Documentation,** Documentacion oficial de Raspberry Pi [online]. Disponible en http://www.raspberrypi.org/help/

[12] **Python Software Foundation: The Python Tutorial.** Página oficial de la fundación Python [online]. Disponible en http://www.docs.python.org/2.7/tutorial

[13] **S. Balasubramanian**, "MavLink Tutorial for Absolute Dummies Part I", Tutorial, Netherlands

[14] **Dronekit**. Página web del fabricante 3DRobotics [online]. Disponible en http://python.dronekit.io/

[15] **Python Course: Tkinter**. Recurso tutorial [online]. Disponible en http://www.python-course.eu/python_tkinter.php

[16] **Python: TCP Communication**. Recurso tutorial [online]. Disponible en: https://wiki.python.org/moin/TcpCommunication#TCP_Communication

[17] Red Hat Enterprise Linux 4. Manual de referencia [online]. Disponible en http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

[18] How to set up Raspberry Pi as a Wifi Access Point,Tutorial [online]. Disponible en:http://xmodulo.com/rasperry-pi-wifi-access-point.html

ANEXO: CÓDIGOS EN PYTHON

Este anexo a la memoria del proyecto contiene los distintos programas realizados en lenguaje Python para llevar a cabo los desarrollos expuestos. Se recomienda la lectura del capítulo 4.4 para poder preparar las placas Raspbery Pi e instalar el software necesario para que los códigos funcionen adecuadamente.

mission_script.py

```
# ------------------------------------------------------------------#
#
# File: mission_script.py
# Created on: 12/2015
#   Author: Sergio Vigorra Trevino
#   e-mail: svigorra@gmail.com
#   TFM: Diseno, montaje y puesta a punto de vehiculos multirrotor
#        para su uso en redes de comunicaciones aereas
#   Description: Programa para ser ejecutado en una Raspberry Pi y
#                cargar una misión en el autopiloto APM. Los
#                waypoints se definen en los arrays Lat y Lon.
#
#------------------------------------------------------------------#

from dronekit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal, Command
import time
import math
from pymavlink import mavutil
import sys

# Coordenadas
Lat = [37.914261, 37.914770, 37.914563, 37.913531]
Lon = [-4.748685, -4.747995, -4.750134, -4.748993]

def download_mission():
    """
    Download the current mission from the vehicle.
    """
    cmds = vehicle.commands
    cmds.download()
    cmds.wait_ready() # wait until download is complete.

def add_mission():
    global Lat
    global Lon
    cmds = vehicle.commands

    print " Clear any existing commands"
    cmds.clear()

    print " Define/add new commands."

    cmds.add(Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, 10))
    cmds.add(Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[0], Lon[0], 11))
    cmds.add(Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[1], Lon[1], 12))
    cmds.add(Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[2], Lon[2], 13))
    cmds.add(Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[3], Lon[3], 14))

    print " Upload new commands to vehicle"
    cmds.upload()


# Conexion con APM
```

```
serial_port_apm = '/dev/serial/by-id/usb-
Arduino__www.arduino.cc__Arduino_Mega_2560_74039313632351105212-if00'
print 'Connecting to vehicle on: %s' % serial_port_apm
vehicle = connect(serial_port_apm, wait_ready=True)

print 'Create a new mission'
add_mission()


# Cerrar objeto "vehícle" antes de salir de cerrar el script
print "Close vehicle object"
vehicle.close()

sys.exit()
```

mission_loader.py

```
# ------------------------------------------------------------------#
#
# File: mission_loader.py
# Created on: 12/2015
#   Author: Sergio Vigorra Trevino
#   e-mail: svigorra@gmail.com
#   TFM: Diseno, montaje y puesta a punto de vehiculos multirrotor
#        para su uso en redes de comunicaciones aereas
#   Description: Programa para ser ejecutado en una Raspberry Pi y
#                cargar una misión en el autopiloto APM. Este
#                programa crea una GUI desde la cual se pueden
#                introducir los waypoints de la misión.
#
#------------------------------------------------------------------#


from dronekit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal, Command
import time
import math
from pymavlink import mavutil
import sys

from Tkinter import*

root = Tk()

def GUI():

    def load_mission():

        global Lat
        global Lon
        global Alt

        Lat = [0,0,0,0]
        Lon = [0,0,0,0]
        Alt = [0,0,0,0]

        #Coordenadas
        Lat[0] = float(latitud_1_text_input.get())
        Lat[1] = float(latitud_2_text_input.get())
        Lat[2] = float(latitud_3_text_input.get())
        Lat[3] = float(latitud_4_text_input.get())

        Lon[0] = float(longitud_1_text_input.get())
        Lon[1] = float(longitud_2_text_input.get())
        Lon[2] = float(longitud_3_text_input.get())
        Lon[3] = float(longitud_4_text_input.get())

        Alt[0] = float(altura_1_text_input.get())
        Alt[1] = float(altura_2_text_input.get())
        Alt[2] = float(altura_3_text_input.get())
        Alt[3] = float(altura_4_text_input.get())


        def download_mission():
            """
            Download the current mission from the vehicle.
            """
            cmds = vehicle.commands
```

```
            cmds.download()
            cmds.wait_ready()

        def add_mission():
            global Lat
            global Lon
            global Alt
            cmds = vehicle.commands

            print " Clear any existing commands"
            cmds.clear()

            print " Define/add new commands."

            cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, 10))

            cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[0], Lon[0],
Alt[0]))
            cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[1], Lon[1],
Alt[1]))
            cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[2], Lon[2],
Alt[2]))
            cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, Lat[3], Lon[3],
Alt[3]))

            print " Upload new commands to vehicle"
            cmds.upload()

        # Conexión con la APM
        serial_port_apm = '/dev/serial/by-id/usb-
Arduino__www.arduino.cc__Arduino_Mega_2560_74039313632351105212-if00'
        print 'Connecting to vehicle on: %s' % serial_port_apm
        vehicle = connect(serial_port_apm, wait_ready=True)

        print 'Create a new mission'
        add_mission()


        # Cerrar objeto "vehicle" antes de cerrar el script
        print "Close vehicle object"
        vehicle.close()


    # Label de titulo
    label_titulo = Label(root, text="Mission Loader 1.0")
    label_titulo.grid(row=0,column=2)

    # Labels de latitud/longitud/altura/waypoint
    label_waypoint = Label(root, text = "Waypoint")
    label_waypoint.grid(row=1,column=0)

    label_latitud = Label(root, text = "Latitud")
```

```
label_latitud.grid(row=1,column=1)

label_longitud = Label(root, text = "Longitud")
label_longitud.grid(row=1,column=2)

label_altura= Label(root, text = "Altura")
label_altura.grid(row=1,column=3)

#Labels de n° de waypoint
label_1 = Label(root, text = "1.")
label_1.grid(row=2,column=0)

label_2 = Label(root, text = "2.")
label_2.grid(row=3,column=0)

label_3 = Label(root, text = "3.")
label_3.grid(row=4,column=0)

label_4 = Label(root, text = "4.")
label_4.grid(row=5,column=0)

# Text inputs

latitud_1_text_input = Entry(root)
latitud_1_text_input.grid(row=2, column=1)
latitud_1_text_input.insert(10,"0")

latitud_2_text_input = Entry(root)
latitud_2_text_input.grid(row=3, column=1)
latitud_2_text_input.insert(10,"0")

latitud_3_text_input = Entry(root)
latitud_3_text_input.grid(row=4, column=1)
latitud_3_text_input.insert(10,"0")

latitud_4_text_input = Entry(root)
latitud_4_text_input.grid(row=5, column=1)
latitud_4_text_input.insert(10,"0")

longitud_1_text_input = Entry(root)
longitud_1_text_input.grid(row=2, column=2)
longitud_1_text_input.insert(10,"0")

longitud_2_text_input = Entry(root)
longitud_2_text_input.grid(row=3, column=2)
longitud_2_text_input.insert(10,"0")

longitud_3_text_input = Entry(root)
longitud_3_text_input.grid(row=4, column=2)
longitud_3_text_input.insert(10,"0")

longitud_4_text_input = Entry(root)
longitud_4_text_input.grid(row=5, column=2)
longitud_4_text_input.insert(10,"0")

altura_1_text_input = Entry(root)
altura_1_text_input.grid(row=2, column=3)
altura_1_text_input.insert(10,"10")

altura_2_text_input = Entry(root)
altura_2_text_input.grid(row=3, column=3)
altura_2_text_input.insert(10,"10")
```

```
    altura_3_text_input = Entry(root)
    altura_3_text_input.grid(row=4, column=3)
    altura_3_text_input.insert(10,"10")

    altura_4_text_input = Entry(root)
    altura_4_text_input.grid(row=5, column=3)
    altura_4_text_input.insert(10,"10")

    # Boton de carga de mision
    cargar=Button(root, text='Cargar Mision', fg="Black", command =
load_mission)
    cargar.grid(row=6, column=2)

GUI()
mainloop()
```

Pi2Pi_Com_TCP_Tests_Client.py

```
# ------------------------------------------------------------------#
#
# File: Pi2Pi_Com_TCP_Tests_Client.py
# Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa que actuará como cliente en una conexion
#                TCP. Script utilizado para realizar pruebas con la
#                transmisión TCP en un PC convencional.
#
#------------------------------------------------------------------#

# Importar módulos necesarios
import socket
import time

# Referencias de tiempo
current_time = time.time()
previous_time = 0


# Datos de la conexión TCP
host = socket.gethostname()
port = 9995           # The same port as used by the server
Buffer_size = 1024

# Conexión TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setblocking(0)

while True:
    current_time = time.time()
    if (current_time - previous_time > 0.3):
        previous_time = current_time
        try:
                print 'Intentando conectar'
            s.connect(('localhost', port))
        except socket.error:
            A = 1
        try:
                data = s.recv(Buffer_size)
            if not data: data = 'NoData'
        except socket.error:
            data = 'NoData'
        except socket.timeout:
            data = 'NoData'
        print data
```

Pi2Pi_Com_TCP_Tests_Server.py

```
# ------------------------------------------------------------------#
#
# File: Pi2Pi_Com_TCP_Tests_Server.py
# Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa que ejerce de servidor TCP. Cuando un
#                cliente se conecte, el servidor le mandara una
#                cadena de caracteres determinada. Script utilizado
#                para realizar pruebas con la transmision TCP en un
#                PC convencional.
#
#------------------------------------------------------------------#

# Importar módulos necesarios
import socket
import time

# Referencias de tiempo
current_time  = time.time()
previous_time = time.time()
start_time    = time.time()

# Datos de la conexión TCP
TCP_IP = ''
TCP_PORT = 9995
BUFFER_SIZE = 1024

# Conexión TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
print 'Esperando un cliente...'
s.listen(1)
print 'Conectado. Dirección:', addr
conn, addr = s.accept()
data = 'Paquete transmitido\n'

while ((time.time() - start_time) < 120):
    current_time = time.time()
    if (current_time - previous_time > 0.3):
        previous_time = current_time
        try:
            conn.sendall(data)
        except socket.error:
            print "Error Occured"
            continue
conn.close()
```

Pi2Pi_Com_Tests_Quad2.py

```
# ----------------------------------------------------------------------#
#
# File: Pi2Pi_Com_Tests_Quad2.py
#   Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa que actuará como cliente en una conexion
#                TCP. Pensado para ir ejecutarse en una Raspberry,
#                en el denominado Quad 2.
#
#----------------------------------------------------------------------#

# Importar módulos necesarios
import socket
import time

# Referencias de tiempo
current_time = time.time()
previous_time = 0


# Datos de la conexión TCP
host = '169.254.6.80'
port = 9995          # The same port as used by the server
Buffer_size = 1024

# Conexión TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setblocking(0)

while True:
    current_time = time.time()
    if (current_time - previous_time > 0.3):
        previous_time = current_time
        try:
                print 'Intentando conectar, Quad 1'
            s.connect(('localhost', port))
        except socket.error:
            A = 1
        try:
                data = s.recv(Buffer_size)
            if not data: data = 'NoData'
        except socket.error:
            data = 'NoData from Quad 1'
        except socket.timeout:
            data = 'NoData from Quad 1'
        print data
#s.close()
```

Pi2Pi_Com_TCP_Tests_Quad1.py

```
# ------------------------------------------------------------------#
#
# File: Pi2Pi_Com_Tests_Quad1.py
#   Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa que ejerce de servidor TCP. Cuando un
#                cliente se conecte, el servidor le mandara una
#                cadena de caracteres invariante. Script para ser
#                ejecutado en una Raspberry Pi en el denominado
#                Quad 1. Necesaria conectividad Wi-Fi entre ambas
#                Raspberrys.
#
#------------------------------------------------------------------#

# Importar módulos necesarios
import socket
import time

# Referencias de tiempo
current_time  = time.time()
previous_time = time.time()
start_time    = time.time()

# Datos de la conexión TCP
TCP_IP = ''
TCP_PORT = 9995
BUFFER_SIZE = 1024

# Conexión TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
print 'Waiting for Quad2...'
s.listen(1)
print 'Conectado. Dirección:', addr
conn, addr = s.accept()
data = 'Data from Quad 1\n'

while ((time.time() - start_time) < 120):
    current_time = time.time()
    if (current_time - previous_time > 0.3):
        previous_time = current_time
        try:
            conn.sendall(data)
        except socket.error:
            print "Error Occured"
            continue
conn.close()
```

Pi2Pi_Com_Quad1.py

```
# ------------------------------------------------------------------#
#
# File: Pi2Pi_Com_Quad1.py
# Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa para ir embarcado en el Quad 1 durante una
#                misión Pi2Pi. Su cometido es grabar un fichero en el
#                que almacena su posición y la posición del Quad 2,
#                en caso de que exista conexión entre ellos.
#
#------------------------------------------------------------------#

# Importar módulos
from dronekit import connect, VehicleMode
import time
import sys
import socket

# Referencias de tiempo
current_time = time.time()
previous_time = 0
start = time.time()

# Datos de la conexion TCP con Drone2
TCP_IP = '169.254.6.80'                          # IP del Drone 2
TCP_PORT = 9999
BUFFER_SIZE = 1024

# Conexión con Drone1 (local)
connection_string = '/dev/serial/by-id/usb-
Arduino__www.arduino.cc__Arduino_Mega_2560_7403931363235170A162-if00'
vehicle = connect(connection_string, wait_ready=True)
print "\nConnecting to vehicle on: %s" % connection_string
time.sleep(5)

# Espera al armado del Drone1 (local) para empezar LOG
while (str(vehicle.armed) == 'False'):
      a = 1

# Drone1 armado, apertura del archivo de datos de vuelo
LOG = open('Datos.txt','w')
LOG.close()
print 'Archivo de LOG abierto'
LOG = open('Datos.txt','a')
LOG.write('Inicio del registro de datos de vuelo:\n\n')
LOG.close()

# Conexión TCP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setblocking(0)

while(str(vehicle.armed) != 'False'):
      # while ((time.time() - start) < 30 ):
      current_time = time.time()
    if ((current_time - previous_time) > 0.3):
        previous_time = current_time
        try:
            print 'He llegado a intentar conectarme'
```

```
            sock.connect((TCP_IP, TCP_PORT))
        except socket.error:
            A = 1 # no hacer nada
            print 'como no recibia continuo'

        Position_local  = str(vehicle.location.global_frame)      # Obtenemos
posicion del drone local
        # Position_local = 'Drone1 en Sevilla'

        try:
            Position_remote = sock.recv(BUFFER_SIZE)
            if not Position_remote: Position_remote = 'NoData'
        except socket.error:
            Position_remote = 'Nodata'
        except socket.timeout:
            data = 'NoData'
            ##        print data

    print 'estamos escribiendo una linea en el log'
    LOG = open('Datos.txt','a')
    LOG.write('Posicion Drone 1: ')
    LOG.write(Position_local)
    LOG.write('    ')
    LOG.write('Posicion Drone 2: ')
    LOG.write(Position_remote)
    LOG.write('\n')
    LOG.close()

# Llegados este punto, la mision ha terminado y drone1 esta desarmado
print "\nClose vehicle object"
#vehicle.close()
sock.close()
sys.exit()

print("Mision Completada")
```

Pi2Pi_Com_Quad2.py

```
# ------------------------------------------------------------------#
#
# File: Pi2Pi_Com_Quad2.py
# Created on: 05/2016
#   Author: Sergio Vigorra Treviño
#   e-mail: svigorra@gmail.com
#   TFM: Diseño, montaje y puesta a punto de vehículos multirrotor
#        para su uso en redes de comunicaciones aéreas
#   Description: Programa para ir embarcado en el Quad2 durante una
#                misión Pi2Pi. Su cometido es funcionar como servidor
#                TCP y enviar a Drone1 su posición GPS continuamente.
#
#------------------------------------------------------------------#

# Importacion de modulos
from dronekit import connect, VehicleMode
import socket
import time

current_time = time.time()
previous_time = 0
TCP_PORT = 9999
TCP_IP = ''
BUFFER_SIZE = 1024

#----------Conexion con la placa APM local----------#

connection_string = '/dev/serial/by-id/usb-
Arduino__www.arduino.cc__Arduino_Mega_2560_74039313632351700011-if00'

# Set `wait_ready=True` to ensure default attributes are populated before
`connect()` returns.
##vehicle = connect(connection_string, wait_ready=True)
print "\nConnecting to vehicle on: %s" % connection_string
time.sleep(5)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connection address:' ,addr
while True:
    print 'Evaluar' #-- # Esta linea sirve de poco
    current_time = time.time()
    if ((current_time - previous_time) > 0.3):
        previous_time = current_time
        GPS_Coordinates = str(vehicle.location.global_frame)
        ##GPS_Coordinates = 'coordenaadas del drone 2'
        try:
            conn.sendall(GPS_Coordinates)
        except socket.error:
            print 'Error Occured'
            continue
conn.close()
vehicle.close()
```